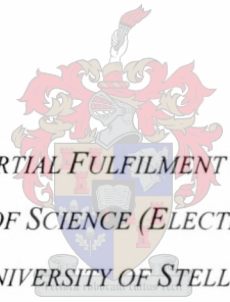
Feasibility of Using an ARM Processor in a Micro Satellite On-Board Computer

Arno Barnard



Thesis Presented in Partial Fulfilment of the Requirements for the Degree of Master of Science (Electronic Engineering) at the University of Stellenbosch

Supervisor: Prof P.J. Bakkes

December 2001

Declaration

I, the undersigned, declare hereby that the work presented in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Arno Barnard Date

Opsomming

Nadat SUNSAT in 1999 gelanseer is en die moontlikheid vir die ontwikkeling van 'n soortgelyke satellite ontstaan het, is areas vir verbetering op bestaande stelsels geïdentifiseer. Die idee om 'n nuwe generasie verwerker met lae kragverbruik as 'n aanboord rekenaar te gebruik, was een moontlikheid wat ondersoek moes word. Die verwerker moes aan riglyne gemeet word wat afgelei is uit die ondervinding wat deur navorsers en studente tydens die SUNSAT projek opgedoen is.

Die tesis dek die proses wat gevolg is om 'n verwerker te kies en die daaropvolgende toetsing om die bruikbaarheid van die verwerker as 'n aanboord rekenaar te bepaal. As deel van die toetsing is die verwerker se spesifikasies teen die verlangde spesifikasies opgeweeg, en 'n evaluasiebord is ontwikkel om die gemaklikheid van die gebruik en programmering van die verwerker te bepaal.

Abstract

After SUNSAT was launched in 1999 and the possibility of developing another micro satellite emerged, areas of possible improvement were identified. Among the suggestions that emerged was the idea of developing an on-board computer for a micro satellite, using one of the new generation low power processors. The processor had to meet guideline specifications derived from the experience gained by researchers and students involved with the SUNSAT project.

This thesis covers the process of choosing one of these processors and the subsequent testing of the feasibility of using it in an on-board computer. The feasibility test included comparing the processor's specifications to those required and designing an evaluation board for this processor to test its ease of use and programmability.

Acknowledgments

I should like to thank the following people who helped me in many different ways: Firstly I should like to thank Professor Bakkes for his belief in me and for his constant support, encouragement and guidance. Secondly I should like to thank Francois Retief, Niki Steenkamp, Heiko Berner, Hans Grobler, Heinrich Venter, Pieter Kotze and Francois Nel for their technical advice and help with various matters pertaining to this thesis. I should also like to thank Cindy Wiggett for her help in proofreading this document, and for her constant moral and emotional support. Lastly, and most important I should like to thank The Lord, for without Him, this work would not have been possible.

Contents

List of Figures

List of Tables

List of Acronyms

ADCS	Attitude Determination and Control System
ALU	Arithmetic Logic Unit
AMSAT	The Radio Amateur Satellite Corporation
ARM	Advanced RISC Machines
BRD	Baud Rate Divisor
CCD	Charged-Coupled Device
CCF	Clock Configuration Field
CISC	Complex Instruction Set Computer
CPU	Central Processing Unit
DIP	Dual Inline Package
DMA	Direct Memory Access
DMMU	Data Memory Management Unit
DRAM	Dynamic Read-And-Write Memory
DSP	Digital Signal Processor
EDAC	Error Detection And Correction
EDO	Extended Data Out
EEPROM	Electrically Erasable Programmable Read Only Memory
EMI	Electromagnetic Interference
EPROM	Erasable Programmable Read Only Memory
ESL	Electronic Systems Laboratory
EVB	Evaluation Board
FIFO	First-In-First-Out buffer
FIQ	Fast Interrupt Request
FLPROM	Fusible Link PROM
FM0	Frequency Modulation Zero
FSK	Frequency Shift Keying
GCC	GNU Cross Compiler
GDB	GNU DeBugger
GNU	GNU's Not Unix
GPS	Global Positioning System
IB	Instrumentation Bus

IC	Integrated Circuit
ICMR	Interrupt Controller Mask Register
ICP	Infrared Communications Port
IHU	Integrated Housekeeping Unit
IMMU	Instruction Memory Management Unit
I/O	Input / Output
IrDA	Infrared Data Association
IRQ	Interrupt Request
ISAC	Intra-Satellite Asynchronous Channel
JTAG	Joint Test Action Group
JVM	Java Virtual Machine
LART	Linux Advanced Radio Terminal
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LEO	Low Earth Orbit
LR	Link Register
LSB	Least Significant Bit
MCP	Multimedia Communications Port
MIPS	Million Instructions Per Second
MPCM	Memory and PCMCIA Control Module
MPE	Microprocessor Engineering Limited
MSB	Most Significant Bit
MSC	Static Memory Control
NRZ-I	Non-Return-to-Zero-1's-Invert
NRZ-L	Non-Return-to-Zero-Level
OBC1	Primary On-Board Computer
OBC2	Secondary On-Board Computer
OS	Operating System
P3D	Phase 3D
PC	Personal Computer
PCB	Printed Circuit Board
PCM	Peripheral Control Module
PCMCIA	Personal Computer Memory Card International Association
PIC	Programmable Interrupt Controller
PLL	Phase Lock Loop

PMCR	Power Manager Control Register
PMU	Power Management Unit
PPCR	Power manager PLL Configuration Register
RAM	Random Access Memory
RB	Read Buffer
RBW	ROM Bus Width
RDF	ROM Delay First access field
RDN	ROM Delay Next access field
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
RRR	ROM/SRAM Recovery time field
RT	ROM Type
RTC	Real-Time Clock
SCM	System Control Module
SDLC	Synchronous Data Link Controller
SDK	Software Development Kit
SEU	Single Event Upset (due to radiation dose)
SNAP	Surrey Nano-satellite Application Platform
SRAM	Static Read-And-write Memory
SSB	SUNSAT Serial Bus
SSIS	SunSpace Information Systems
SSP	Synchronous Serial Port
SSTL	Surrey Space Technology Limited
SUNSAT	Stellenbosch University Satellite
TCMD	Tele-command
UART	Universal Asynchronous Receiver/Transmitter
UDC	USB Device Controller
USB	Universal Serial Bus
UTCR	UART Control Register
UTSR	UART Status Register
VDD	SA1100 core supply voltage
VLSI	Very Large Scale Integration
WB	Write Buffer

1 Introduction

In 1991 the development of the Stellenbosch University Satellite (SUNSAT) was commenced at the Electronic Systems Laboratory (ESL). SUNSAT was developed using low cost design techniques and components. The SUNSAT project created new interest in the development and research of satellite systems. With the possibility of more micro satellite programs to follow the very successful SUNSAT project, it is important that researchers continue to search for newer and more effective satellite systems to use in such programs. Lessons learned during the SUNSAT project and consideration of future possibilities revealed the need to find and test a new generation, low cost, low power microprocessor for use as a satellite on-board computer.

1.1 Background

The project had the goal of designing and building a low-cost and reliable micro satellite, using commercial grade components. Space-qualified and radiation-hardened components are very expensive but also very reliable. Commercial grade components are very inexpensive but, unfortunately, less reliable. It was decided to try and offset the reliability problem by building more redundancy into the system. This approach had the result of greatly reducing the cost of the satellite while still maintaining a high reliability factor. SUNSAT's Primary on-board computer (OBC1) is an 80188 Intel processor-based system, while the Secondary on-board computer (OBC2) is an 80386EX Intel processor-based system. The reasons for using these specific processors are described in the masters' theses of H. Grobler[3] and N. Goosen[2].

After SUNSAT was launched, problems relating to the 80386EX computer were discovered. There were differences in the behaviour of the engineering model and the flight model, which limited the debugging process on the OBC2 flight model. The 80386EX is a very robust processor, manufactured with older technologies. This results in relatively big silicon structures on the die, which make the processor less susceptible to harmful radiation. This is a positive attribute for processors used in space applications. The 80386EX processor does, however, have some drawbacks. Compared to new generation processors, its power consumption to performance ratio is much lower. A further

disadvantage of the 80386EX is that the manufacturing of the processor might be discontinued in the near future.

Several new generation processors were released on the market in the last ten years. Compared to the 80386EX, these processors might be better suited to on-board computers.

1.2 Outline

- Chapter 1 introduces the subject of the thesis, gives an overview of the background of the SUNSAT project and an outline of the document.
- Chapter 2 provides an account of the literature study done to choose a suitable processor.
- Chapter 3 describes the detail functionality of the processor chosen and the design process of the evaluation board.
- Chapter 4 gives an overview of the software environment of the processor and the details of the development of the system software.
- Chapter 5 discusses the test and debugging procedures performed on the evaluation board.
- Chapter 6 presents the conclusions drawn from the study and gives recommendations with regard to improving the system and possible future development that can be undertaken

2 Literature study

2.1 Previous work done on On-Board Computers

This section will describe the work and conclusions (relating to the design, development and evaluation) of N. Goosen [2] and H. Grobler [3] on SUNSAT's OBC1 and OBC2. A system diagram of the SUNSAT core architecture is shown below to put the following descriptions into context.

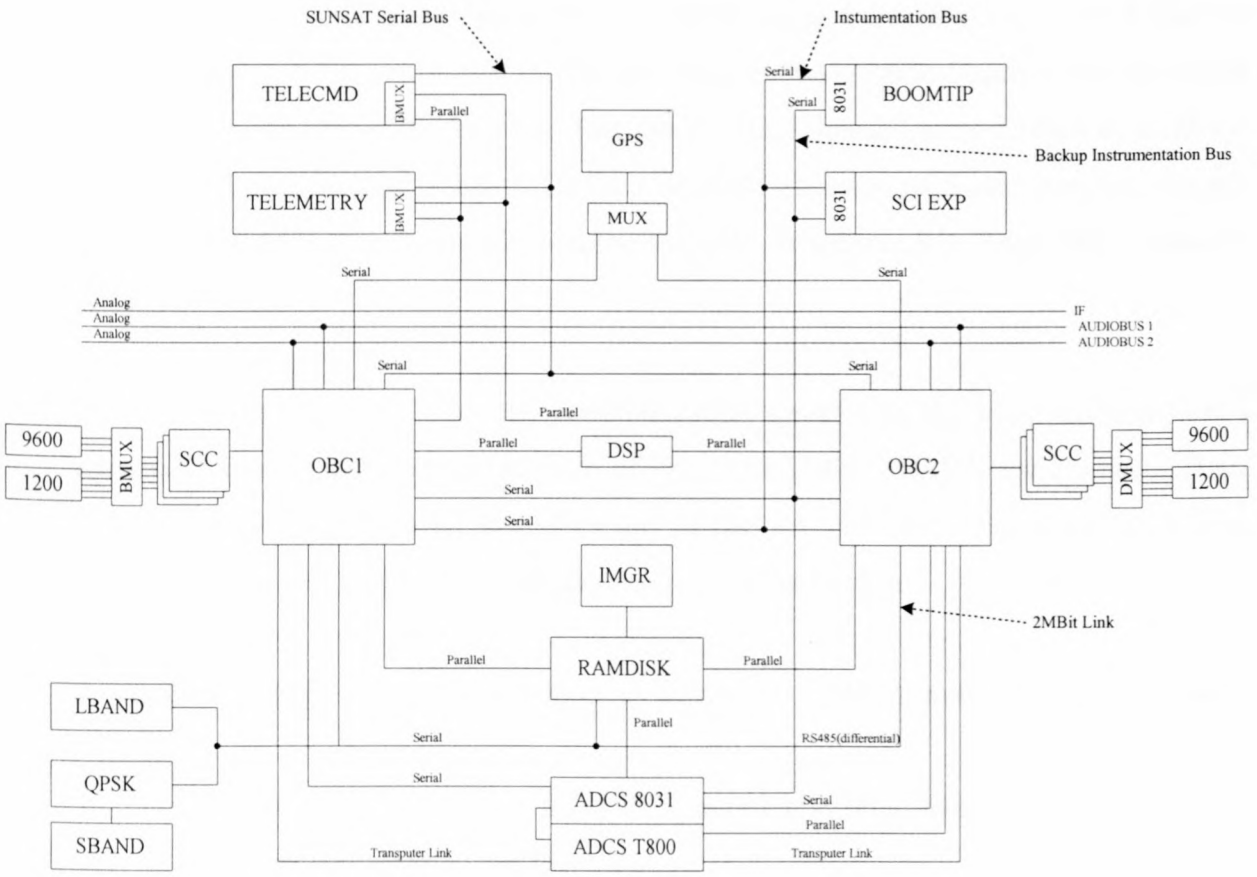


Figure 2-1 Diagram of SUNSAT core architecture

OBC1, as mentioned earlier, is an 80188 Intel processor-based system. The processor had a 8-bit external bus (16-bit internal), idle and power-down modes, 8259 compatible interrupt controller, timer unit, chip select unit, 4-channel Direct Memory Access (DMA) unit, Universal Asynchronous Receiver/Transmitter (UART), refresh control unit and watchdog unit. This processor had performed very well in the space environment up to the end of

SUNSAT's life (February 1999 to February 2001). During this time, no Single Event Upsets (SEUs) or other radiation-related errors where detected inside the processor. This ensured the future use of the processor in the primary OBC for the next ESL micro-satellite project. The OBC1 address and data bus architectures are central star-type architectures, i.e. the busses spread out from the processor to all the necessary components, with added buffers in certain branches to prevent overloading. In most OBC designs where only one processor is used, this is the desired architecture. The busses spread out to about 20 different subsystems or components.

OBC2's processor, the 80386EX, mostly had the same peripherals, within or connected to it, as the 80188. There are some exceptions, however. One of these is the direct link to the Attitude Determination and Control System (ADCS) in case the T800 transputer failed. To enable OBC2 to take over all the T800's functions, an 80387 Math-Coprocessor was added to its design. Another difference was that the 80386EX needed more buffers to drive the address and data busses. Finally, OBC2 had different sizes of Static Random Access Memory (SRAM) and Flash Electrically Erasable Programmable Read Only Memory (EEPROM) connected to it.

In order to communicate with the communication systems in the satellite, both OBCs required six serial channels in the form of three μPD72001 USART devices each. Their outputs could be multiplexed to service any of the 7 x 1200 baud Frequency Shift Key (FSK) modems and 3 x 9600 baud modems on board the satellite.

A number of Intra-Satellite Asynchronous Channels (ISACs) were used in SUNSAT, including:

- The SUNSAT Serial Bus (SSB), running at 9600 baud with a 8N1 (8 data bits, no parity, 1 stop bit) frame
- The Instrumentation Bus (IB), running at 19200 baud with a 8P1 (8 data bits, parity, 1 stop bit) frame
- The Global Positioning System (GPS) interface ISAC, running at 19200 baud with a 8N1 (8 data bits, no parity, 1 stop bit) frame

In addition to these serial communication channels the OBCs also had a number of parallel interfaces, i.e.:

- The Internal Tele-command (TCMD) Interface
- The Internal RAMDISK (64MB SRAM tray) Interface
- The Internal IMAGER (single line CCD camera) Interface
- The Internal Digital Signal Processor (DSP) Interface
- The Internal ADCS Interface
- The Internal T800 Transputer Interface

All these parallel interfaces were connected to the data and address busses. Communication to and from them was controlled by the OBCs.

Most of the above-mentioned systems / components communicate service requests to the processor by means of interrupts. This means that a very high interrupt load was put on the OBCs. To alleviate this load, a Programmable Interrupt Controller (PIC) was added, which created enough interrupt channels to work with. This was deemed the best solution to the problem of servicing the large number of sub-systems.

Knowledge of how these two computers were put together (and why) helped to develop an idea of what a future OBC design would be compared to. There are numerous advantages and disadvantages to the implemented designs of OBC1 and OBC2. How a new processor, chosen later in this report, would possibly result in changes to the designs, will be discussed in Chapter 6.

2.2 Processor specifications

Processor specifications have to be drawn up before the search for a new processor can begin. Usually the functional specifications for an on-board computer design are determined by the satellite system requirements. Due to the fact that a specific satellite system is still undefined, the process of drawing up specifications in the conventional way is impossible. It is however possible to draw up a set of guidelines, partly from previous OBC designs and experiences as described in Section 2.1, and partly from ideas of what future systems might look like. The processor guidelines that follow are mostly independent of the detailed system design.

- To optimise the data transfer efficiency, the processor should have a full 32-bit architecture.
- The processor must be a commercial grade processor, not radiation hardened, to minimize costs.
- Radiation effects should be minimised and the processor should be manufactured using at least a 0.5µm manufacturing process.
- The processor must have no CACHE memory, which is very susceptible to radiation. If it is present, it must be possible to disabled the cache.
- A 5-volt supply voltage would be preferred to minimise the probability of errors on the address and data busses.
- The processor should perform at least 5 Million Instructions Per Second (MIPS) while running at 20MHz to be able to handle the estimated workload (based on the software used on SUNSAT).
- To minimise the possibility of data corruption, the processor must be able to interface with an Error Detection And Correction (EDAC) module or have an on-chip EDAC unit.
- The processor should be able to operate in a Low Earth Orbit (LEO) satellite without the need to add a radiation shield, which could be costly and add extra, unwanted, weight to the OBC design.
- To make the development cycle shorter and to keep development costs low, the processor should have good software support, which requires no (or inexpensive) licence fees.
- Due to the limited power available (and to limit heat generation) on board the satellite, the processor should have low power consumption (less than 1 Watt).

These guidelines were drawn up with consideration of the experience gained through the SUNSAT project.

2.2.1 Possible choices for a processor

There is a large number of new generation processors on the market. A search for a possible processor to fit the guidelines mentioned previously, had to be executed. The Internet was used extensively. The first guideline states that the processor should have a 32-bit architecture. Following the search based on this criterion, a distinction was made between Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computer (RISC)-based processors. The focus was mainly on RISC based processors

because of their low power consumption. A complete list of the RISC processors found is shown in Appendix A.

These processors are built with a 32-bit architecture although some of them do not have full 32-bit data bus interfaces. Four processor families are listed: ARM, MIPS, PowerPC and Super-H. From these lists a few processors could be identified as possible choices. Among them were the ARM7100 processor, the BUTTERFLY processor, the SA1100 processor and the PR31500 processor. The ARM7100 processor can run from a 5V supply and has a full 32-bit architecture. It also has very low power consumption. Unfortunately the available information suggests that the processor can only perform at 18.4 MIPS. The supplier does not have the processor in stock and a large minimum order quantity is required to order any. With regard to the BUTTERFLY processor, its datasheet suggests that it is a very good candidate for use in this study, but none of the suppliers had any information on the availability or price of the processor. The PR31500 processor unfortunately uses only a 3V power supply and was also ruled out due to a lack of information on price and availability from suppliers. The SA1100 processor, however, is available on order and its datasheets suggest that it is a possible option. This processor warranted a closer look.

2.3 The StrongARM SA1100 processor

To ensure that the SA1100 would meet the specifications set out in Section 2.2, the datasheet of the processor has to be examined. A description of the SA1100 processor core and its peripherals will now follow.

Figure 2-2 shows the block diagram of the SA1100 processor as in [4]. The processor core is Intel's ARM SA-1 core. Connected to this core is a 16Kbytes Instruction Cache (ICache) and an 8Kbytes Data Cache (DCache), which in turn are connected to the Instruction Memory Management Unit (IMMU) and the Data Memory Management Unit (DMMU). Both caches have 32-way set associative organization, and both can be enabled or disabled by software. Both are disabled after hardware reset. The core is also connected to a Write Buffer (WB) and a Read Buffer (RB) to improve system performance under certain conditions (see [4] p.6-5 to 6-6). The WB and RB can be enabled or disabled by software (WB disabled after reset).

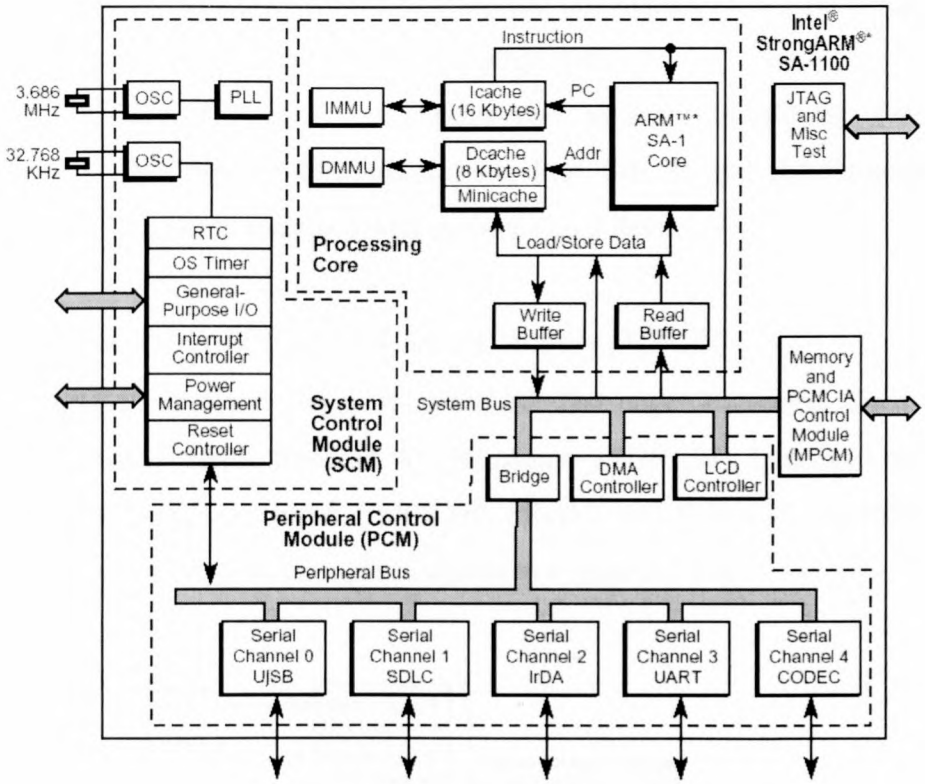


Figure 2-2 SA1100 Block Diagram

The processing core connects via the system bus to the Peripheral Control Module (PCM) and the Memory and Personal Computer Memory Card International Association (PCMCIA) Control Module (MPCM). It is further connected to the System Control Module (SCM) and the Joint Test Action Group (JTAG) and miscellaneous test unit.

2.3.1 Peripheral Control Module (PCM)

The PCM includes the following peripherals:

- *Liquid Crystal Display (LCD) Controller* – Supports single/dual panel display up to 1024 x 1024 pixels, using one of three display types (Passive colour, Active colour and Passive monochrome)
- *DMA Controller* – Six independent channels to service any of the serial ports; data transfers are performed between the port serviced and memory
- *Serial Port 0* – Universal Serial Bus (USB) Device Controller (UDC)
- *Serial Port 1* – Combination of Synchronous Data Link Controller (SDLC) and UART, can be operated in either one of the modes or both
- *Serial Port 2* – Infrared Communication Port (ICP), operates at half-duplex and supports direct access to Infrared Data Association (IrDA) compliant transceivers

- *Serial Port 3* – General-purpose, full-duplex UART port, supports much of the functionality of the 16550 protocol
- *Serial Port 4* – Contains two separate full-duplex serial interfaces, Multimedia Communications Port (MCP) and Synchronous Serial Port (SSP) to interface with various serial devices

2.3.2 Memory and PCMCIA Control Module (MPCM)

The MPCM interface supports standard fast-page Extended Data Out (EDO) asynchronous Dynamic Random Access Memory (DRAM), burst and non-burst Read-Only Memory (ROM), Flash EEPROM, SRAM, and PCMCIA expansion devices. A maximum configuration example is shown in Figure 2-3 [4].

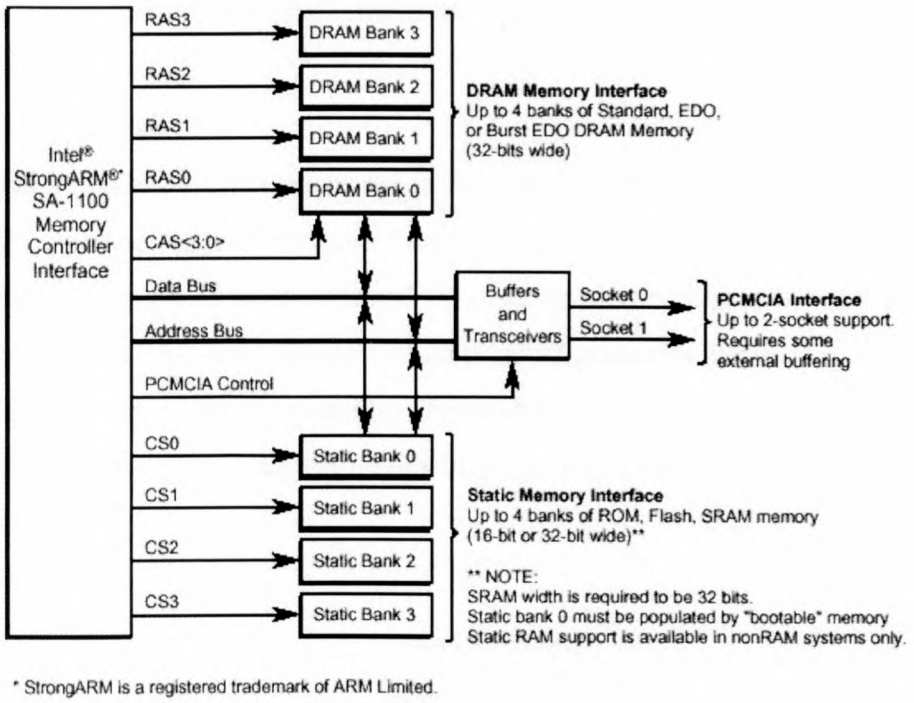


Figure 2-3 General Memory Interface Example

The figure shows how different types of memory can easily be connected to this processor. It is clear, from this example, that a memory system like the one on SUNSAT, using SRAM, Flash RAM and EPROM, can easily be implemented with the system.

Four chip selects are also available to use for on-board memory selection, as well as for use by other peripherals.

2.3.3 *System Control Module (SCM)*

The SCM module contains the following units:

- Two crystal oscillators – 3.6864Mhz oscillator and 32.768kHz oscillator
- Internal phase-lock loop circuit (PLL) – generating internal 48MHz clock and core clock of 59-200MHz
- Real Time Clock (RTC) – general purpose real-time reference clock
- Operating System (OS) Timer – including watchdog timer
- Power Management Unit (PMU) – controls transitions between run, idle and sleep modes
- Interrupt Controller – can handle up to 32 Fast Interrupt Requests (FIQs) and up to 32 Interrupt Requests (IRQs), all from different sources
- Reset Controller – managing the various reset sources

2.3.4 *Power Saving*

A positive feature of this processor is the very low power consumption. The datasheets specify a power usage of no more than 0.5 Watts for a processor running normally. This is very low compared to the 80386EX processors that consumed up to 1.375 Watts. For an application like this, where the amount of power is limited, it is indeed a very attractive feature. The processor has two power saving modes, i.e. idle mode and sleep mode. The following diagram shows how the processor enters and exits the different modes [4].

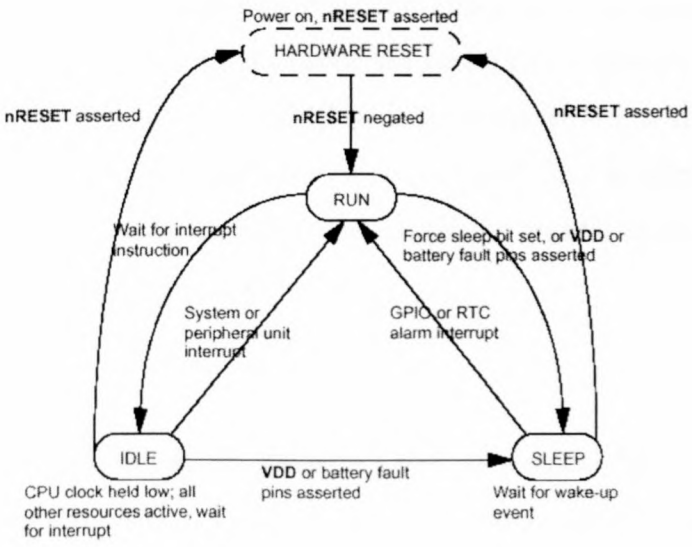


Figure 2-4 Transitions between modes of operation

When the processor is in *run* mode, it can only be set to *idle* mode through a software instruction sequence. In *idle* mode the Central Processing Unit (CPU) clock is stopped. Because the SA1100 is fully static, all the information about the CPU state is saved. All the other systems units and peripheral units (including PLL) are fully operational during *idle* mode. While in *idle* mode, it waits for one of three events to occur. The first is when the nRESET signal is asserted, which will cause a hardware reset to occur before returning to *run* mode (all processor states and unsaved information is lost). The second option is that a system or peripheral interrupt occurs. This will cause the processor to come out of *idle* mode quickly and continue with the processing from exactly the same point where it stopped. *Idle* mode is handy in circumstances where battery power needs to be conserved, while the system is still ready to handle any interrupts that may occur. The third way for the processor to get out of *idle* mode is when the VDD_FAULT or BATT_FAULT signals are asserted by an external power supply. This will cause the processor to go into *sleep* mode. The maximum power consumption in *idle* mode is specified to be about 85mW [4].

In *sleep* mode an internal reset is applied to the processor, the PWR_EN signal is negated by the SA1100, signalling the external power supply that the 2.0V VDDI supply can be disabled. Because only the 3.3V Input / Output (I/O) supply must remain, the maximum amount of power can be saved. The processor can enter *sleep* mode in one of two ways. One is by forcing the sleep-bit in the Power Manager Control Register (PMCR) via software, the other by applying a VDD_FAULT or BATT_FAULT signal to the processor from the external power supply. There are two ways to exit *sleep* mode. If the nRESET signal is asserted, the processor performs a hardware reset, in the same way as exiting from *idle* mode. A GPIO or RTC interrupt alarm is the second set of events that can cause the processor to exit *sleep* mode. This is affected in an ordered sequence (see [4], pages 9-27 to 9-30). The maximum current drawn in *sleep* mode is specified to be no more than 50μA [4].

2.3.5 Memory map

The memory map determines the possible sizes of memory blocks so that efficient programming can be done. It also determines how memory structures can be implemented in hardware. It, therefore, is important to consider the memory map of the

SA1100 when designing a system like this. A diagram of the SA1100 processor's memory map is shown in Figure 2-5.

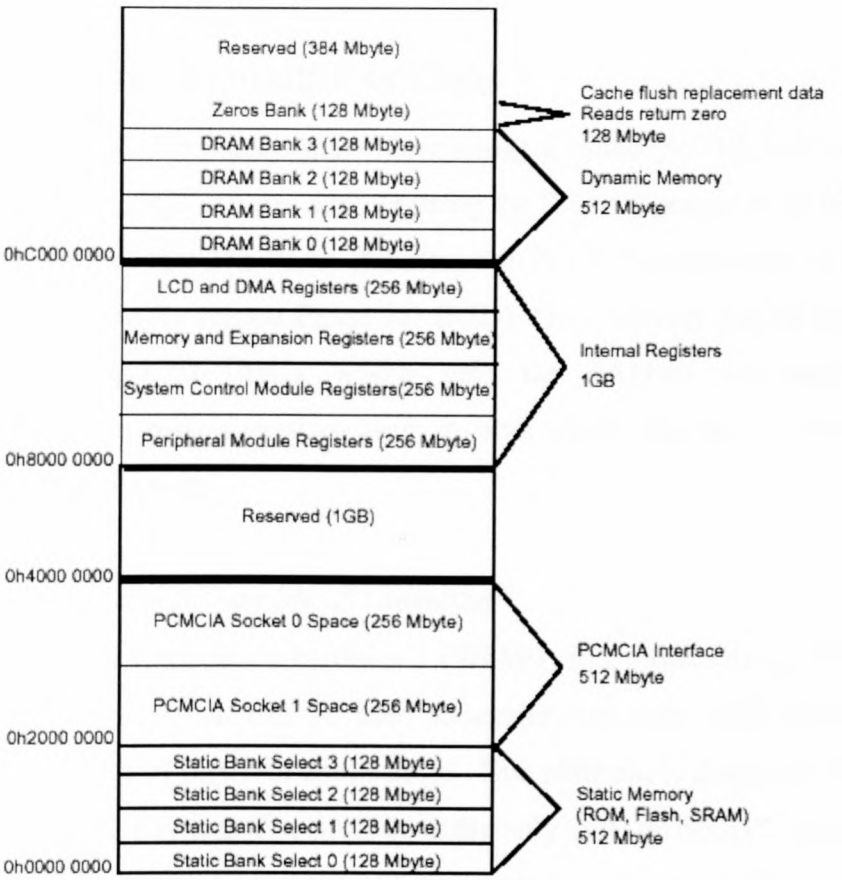


Figure 2-5 SA1100 Memory map

Figure 2-5 shows that the nCS0-nCS3 signals are mapped to physical memory as follows:

- nCS0 maps to 0h0000 0000 - 0h07FF FFFF
- nCS1 maps to 0h0800 0000 - 0h0FFF FFFF
- nCS2 maps to 0h1000 0000 - 0h17FF FFFF
- nCS3 maps to 0h1800 0000 - 0h1FFF FFFF

There are 128M bytes of memory space available for each nCS signal. The four nCS signals can be used to select the different types of static memory that should be used in this study (i.e. SRAM, Flash memory and EPROM). This is more than enough memory space for any OBC type design that typically uses no more than 16M bytes (because the memory ICs take up a lot of Printed Circuit Board (PCB) space) of any kind of memory. If larger memory blocks are required, they can be implemented in the form of a

RAMDISK (a whole PCB tray on its own). The mapping for the PCMCIA and DRAM memory is not being evaluated in this study, since it is very unlikely that these types of memory will be used in satellite systems in the near future.

2.4 SA1100 in other satellite systems

Internet searches showed that Surrey Space Technology Limited (SSTL), a British satellite manufacturing company, built a nano-satellite using the SA1100 processor in the on-board computer. It also showed that The Radio Amateur Satellite Corporation (AMSAT) built a new communication satellite named Phase 3D (P3D). On P3D they put an experimental Integrated-Housekeeping-Unit (IHU), which uses the SA1100 processor. A brief description of how the processor was used in both cases (limited by the available information) will now be given.

2.4.1 StrongARM SA1100 on SNAP1 satellite

Surrey Nano-satellite Application Platform 1 (SNAP1) is a project designed to provide low-cost small satellites that can be used to rendezvous with other satellites or in formations for a range of different applications. This platform is designed to be able to carry a customer built payload. The SA1100 is the only OBC on SNAP1. It performs, as far as can be determined (see [18]), all the housekeeping, as well as the ADCS computing functions. This makes the SA1100 a mission critical component in SNAP1, showing that SSTL has a great deal of trust in the processor and their design. No information is available on the structure of the system architecture. The only facts that have been given are that it is running at the maximum clock speed of 220MHz and uses 2Mbyte Flash memory and 4Mbyte double bit per byte EDAC memory.

2.4.2 StrongARM SA1100 on P3D satellite

The Radio Amateur Satellite Corporation's (AMSAT) Phase 3D (AO-40) satellite was launched 16 November 2000 aboard an Ariane 5 launcher from Kourou, French Guiana. The satellite carries a secondary Integrated-Housekeeping-Unit (IHU-2) built around the SA1100 processor. This system was built as a possible future upgrade from the COSMAC-1802-based flight computer (IHU), used in previous AMSAT satellites. The COSMAC is an 8-bit processor and is becoming obsolete. After numerous processors were screened, AMSAT decided to use the SA1100 system as a technology-proving experiment on-board Phase 3D.

The block diagram of the IHU-2 is shown in Figure 2-6. It was decided that, for optimal use of the SA1100, 128K x 32-bit hardware-protected memory (EDAC) and 8 megabytes (2M x 32-bit) of software-protected memory should be added. This was due to the fact that a large block of protected memory is not power efficient. The software-protected memory is basically used for storing non-critical image data from the on-board Charged-Coupled Device (CCD) camera, while all the critical program code is stored in the hardware-protected memory. If the latter should become corrupted, new software can be uploaded from the ground.

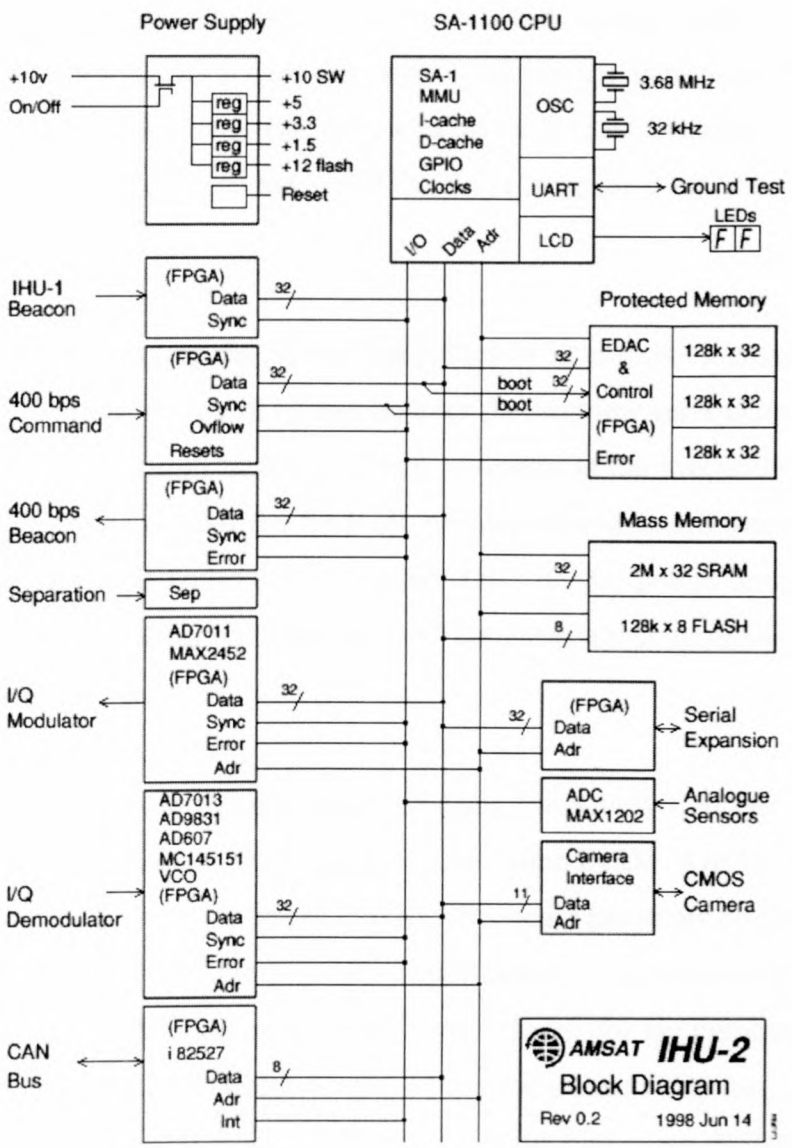


Figure 2-6 IHU-2 Block Diagram

Details about the sub-systems on the IHU-2 are explained in [16]. Although these details had some influence on how certain sections of the evaluation board described in Chapter 3 were designed, it is unnecessary to repeat their descriptions in this report.

2.5 Conclusions

Reviewing the guidelines set for the processor shows that the SA1100 complies with most of them. The only exception is the use of the processor in LEO satellites, which is yet to be determined. The processor, however, has no floating-point unit. This makes it harder to program and less suitable for intensive mathematical work. The SA1100 is manufactured with the use of fully static 0.35μm technology, which is currently one of the biggest manufacturing processes still used for this type of Very Large Scale Integration (VLSI) devices. This is an important factor with regard to how radiation affects the processor. The larger the process, the smaller the chance that radiation charge build-up can bridge silicon gaps, and that a high radiation dose can cause breakthrough damage. Unfortunately there is no information available on any radiation tests on the SA1100 processor. However, the fact that two prominent institutions have used the SA1100, made it easier to decide to use it in this study.

Intel is currently promoting the use of this processor in embedded environments and it is assumed that the processors will be available for the next few years at least. Another factor that influences the decision to use a certain processor is the availability of development software for it. In the case of the SA1100, development software that runs on the Linux operating system and is under the GNU's Not Unix (GNU) public licence is available. A full set of tools including an Assembler, C compiler, cross compiler and debugger is available free of charge.

The next consideration is the possibility of using the SA1100 on one of the evaluation boards currently available. One board is the StrongARM SA1100 PowerBoard by Microprocessor Engineering, Limited (MPE). This board has many on-board peripherals to facilitate development with the SA1100. Unfortunately it has drawbacks. Firstly, it uses Dynamic RAM (DRAM) on-board, which is not needed for the purposes of this study. It also does not have SRAM (which is one of the components that has to be tested). The biggest problem with using this board, however, is the high price (R5600), which exceeds the budget set out for this study. One or two other evaluation boards referred to in searches were either not available anymore or no pricing information could be obtained.

A board that could possibly be used is the Linux Advanced Radio Terminal (LART) board developed by a research group at Delft University of Technology (TU Delft) in the Netherlands. It is basically a whole computer, designed to run a Linux kernel. This board's

design and development software is available free of charge, the only problem being that the design is fixed with little or no means to expand the board. This makes it difficult to test the low-level performance of the SA1100 processor.

Given the problems with the boards that have been mentioned, it was decided to build a new evaluation board. This would provide an opportunity to research the interfaces and the processor extensively, and being able to expand the system in various ways. This approach provides a good solution at a much lower cost. Chapters 3,4 and 5 will describe the process used to achieve this objective.

3 Design of the evaluation board

3.1 Design overview

The most significant factor that determines the architecture and design of a computer system is its functional specification. Currently, there is no available functional specification in which the processor, as mentioned in Chapter 2, will be used. This is also the first time that a processor of this kind (StrongARM) will be used in an evaluation board design in the ESL's academic environment. Detailed information on how the processor was used in other systems, by other institutions, is limited. It is also not useful to design a board to test only one or two specific functions, in which case a whole new board must be designed when other functionalities of the processor itself, cannot be tested.

A decision was taken to design a very flexible evaluation board. The flexibility would ensure that the board:

- was easy to use
- was inexpensive to build
- could be used together with different external bus architectures
- could be used together with different sub-systems
- could be used in different configurations

Keeping the design objectives in mind, the following design specifications were set:

- The board had to be as simple as possible, while still having full functionality of the processor
- To optimise board space only minimal memory should be on the board
- The board should have adequate connection and test pins to facilitate future development and testing easier
- To facilitate hardware debugging and development, development aids such as tester LED's and dipswitches should be built in on the board
- All processor pins used for special purposes, i.e. power management, clocks, debugging hardware and control lines, should be adjustable between either a set value or external control, to simplify configurations

- Data and address lines should all be available to external modules such as memory boards, to simplify expansion designs
- The board should be able to provide power to external modules that might be added later

With a set of design specifications in place, a closer examination of the processor was needed before the functional design of the evaluation board could be done.

3.2 Block diagram and functional design

Figure 3-1 shows the block diagram of the proposed system. It is clear that the SA1100 processor is the core of this board. It was decided that only minimal peripherals should be present on the board, to minimize cost and design time. Only minimal EPROM, Flash and SRAM memory is present. The diagram shows that the central bus architecture was used on the board, due to the fact that it was only a first iteration to test the SA1100 on low-level performance. Using a ring bus or spread bus architecture was unnecessary (although possible in future designs).

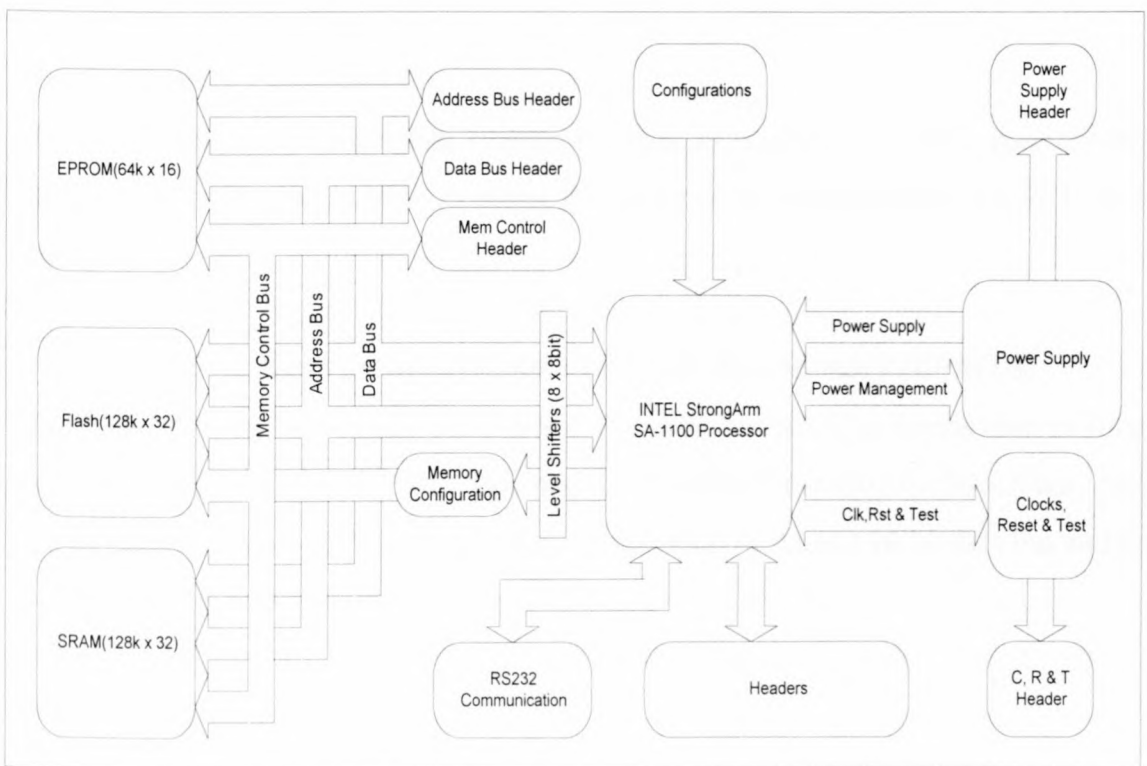


Figure 3-1 SA1100 Evaluation Board Block Diagram

Descriptions of the units in the diagram are given in the following sections.

3.2.1 SA1100 Processor

This is the Intel StrongARM SA1100 32-bit RISC processor as described in chapter 2. The processor acts as the central point of the star-type central bus architecture.

3.2.2 Level Shifters

The level shifters are 74LVXC4245WM 8-bit devices that have a dual supply to "shift" or convert the bus voltages from 3V to 5V levels and vice versa. It was necessary to include these devices in the design since the processor I/O voltages are 3V compatible, while all the memory used on the board are 5V I/O compatible devices. If it is decided in the future that 3V memory will be used in designs for use in satellites, these devices will become redundant. In this design they provide an extra functionality due to the fact that they also serve as buffers for the address and data busses as well as certain memory control lines. The 74LVXC4245WM is rated to source/sink up to 24mA on its output pins, where the SA1100 processor is rated to source/sink only 2mA per pin. This means that no extra buffers had to be used to make sure that the processor does not source/sink too much current.

3.2.3 Memory Configuration

This is a group of jumpers that enables the user to configure the boot memory into different banks, i.e. to be able to select the banks of memory used by the SA1100 to boot from.

3.2.4 Erasable Electrical-Programmable Read Only Memory (EPROM)

The EPROM is an M27C1024-10F1 (64k x 16bit) device. It has an access time of 100ns and temperature range of 0-70°C. This makes it suitable for storing the boot loader code for the evaluation board. This device was chosen because it has a 16-bit data bus and the DIP package is readily available.

3.2.5 Flash memory

This consists of two M29F200B-90M1 (128k x 16-bit) EEPROM ICs (also known as Flash memory). These surface mount ICs were chosen instead of the 8-bit wide versions because they take up less board space. The 32-bit wide version would have been ideal, but is not readily available.

3.2.6 Static Random Access Memory (SRAM)

The SRAM consists of four Hitachi HM628128LFP7 (128k x 8-bit) SRAM ICs. This is the same type of SRAM used on SUNSAT and on a satellite project by Sunspace Information Systems (SSIS). Like the Flash memory and the SA1100 processor these ICs are also surface mounted devices.

3.2.7 Power Supply Unit

The unit consists of the following three voltage regulators:

- One L7805CV IC that provides a fixed 5V supply to the board
- Two LM317 ICs that are individually adjusted to provide a 3.3V and a 2V supply to the board

3.2.8 Clocks Reset and Test

The clock section consists of the 32.768KHz and 3.6864MHz crystals connected directly to the SA1100. The reset section consists of an RC circuit that allows the oscillators enough time to stabilize after hardware reset. The test section consists of a connection header to the JTAG pins on the SA1100, to ensure that debugging can be done easily.

3.2.9 RS232 Communication block

This block consists of two RS232 transceivers connected to the four serial ports of the SA1100. These are 3.3V compatible devices. There is therefore no need to level-shift these lines. This block provides RS232 signals on a header so that direct serial interfacing with the PC or other RS232 devices is possible.

3.2.10 Configurations block

This block consists of a series of jumpers. With these jumpers several signals can either be configured to pre-determined values or be managed by an external board/system. This ensures that easier debugging can be done along with modular development on the system.

3.2.11 Header blocks

Various header blocks are shown in the block diagram. All the headers have the same function - to give easy access to most of the signals on the board. This will aid the

development and connection of daughter boards, which could be added later to extend the functionality of the whole system.

3.3 Detailed design

This section will describe the detailed design of the evaluation board. The schematics of the design, executed in Design Explorer 99 SE (product of Protel International Limited), are shown in Appendix B.

3.3.1 System Clocks

This was the first section to be designed. It represents a simple step, as it only involves connecting the two crystals to the oscillators input pins on the SA1100. The two crystals used are a 3.6864MHz crystal and a 32.768KHz crystal. Figure 3-2 shows the block diagram of the clock structure [4].

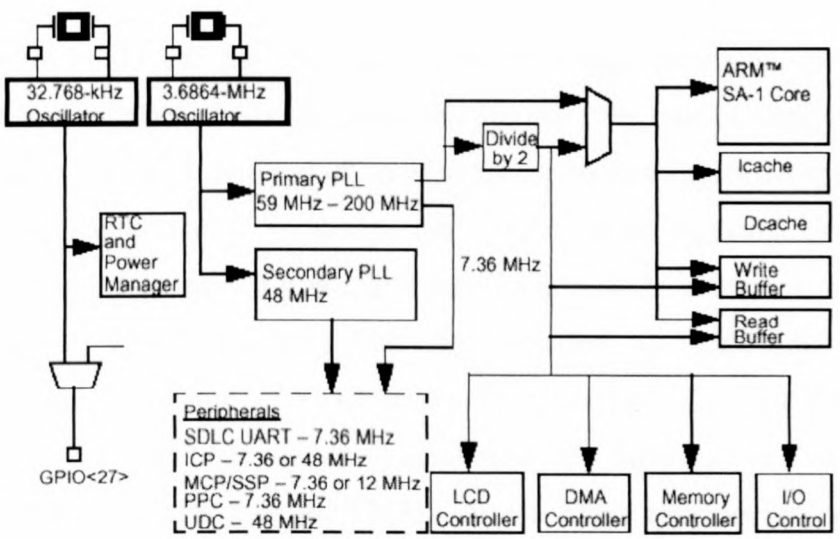


Figure 3-2 Clock distribution in the SA1100

The 3.6864MHz crystal-oscillator is used for the primary internal Phase Lock Loop (PLL) of the SA1100 that generates the internal core clock for the processor. This clock is divided by 2 and then used for the Memory Controller, DMA controller, LCD controller, I/O control, write buffer, read buffer and the instruction cache. The primary PLL also generates a 7.36MHz clock that feeds the SDLC/UART, ICP, MCP/SSP and PPC units. The secondary PLL generates a 48MHz clock that feeds two peripherals, i.e. the UDC unit and the ICP unit. The 32.768KHz crystal-oscillator feeds the Real-Time Clock (RTC) and the power manager clock unit.

3.3.2 Power Supply

The evaluation board needs three fixed voltages to supply power to all the components:

- 2V supply, for the SA1100 core
- 3.3V supply, used for the SA1100 I/O ports and for one of the supplies of the level shifter IC's
- 5V supply, to power all the memory components and the second supply of the level shifters.

All these voltages are also put on headers, so that daughter boards can be supplied from the evaluation board and to make access to measurement points easier.

The LM317T adjustable-output-voltage regulators were used to generate the 3.3V and 2V supplies. In the case of the 2.0V supply, an adjustable regulator was used because no 2.0V fixed voltage regulators could be obtained from any local supplier.

Compared to R4.00 for the adjustable LM317 voltage regulator, the fixed 3.3V regulator that is available costs approximately R60.00. The cost of using the LM317 is also affected by the extra time and effort used for the more complex design. Comparing the difference in time used to design the fixed voltage regulator circuit and the adjustable voltage regulator, it was found that the adjustable voltage regulator did not take significantly longer to design. Therefore the effect of design time costs can be ignored. Another cost factor is the PCB area used by the LM317 circuit compared to the area used by a fixed voltage regulator circuit. The only difference in PCB area is that used by the biasing resistors in the LM317 circuit. These resistors use approximately $0.5cm^2$ (0.5cm x 1cm) board area compared to the total board area that is approximately $215.28cm^2$ (11.7cm x 18.4cm). The resistors therefore use approximately 0.2137% of the total board area, which, when converted into manufacturing costs, is approximately R3.94 (total manufacturing cost of board is approximately R1700.00). For the 3.3V supply it was, therefore, decided to also use an adjustable voltage regulator.

The LM317 was used in a fixed configuration prescribed by the manufacturers, as shown in Figure 3-3.

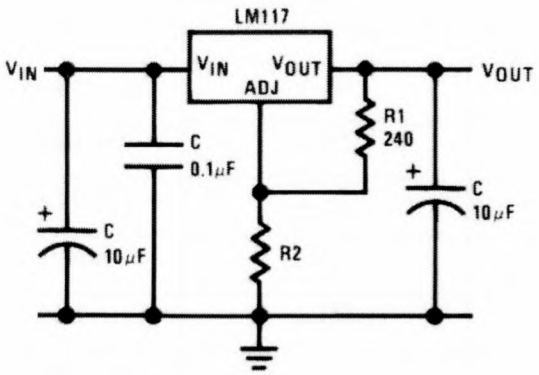


Figure 3-3 LM317 Power regulation circuit

Two resistors determine the output voltage for the regulator. The equation governing this output is

$$v_{OUT} = v_{REF}\left(1+\frac{R2}{R1}\right)+I_{ADJ}R2$$

With V_{REF} = 1.25V, R2 = 240Ω (from datasheet) and choosing V_{OUT} as 2V, the value of R1 is calculated as being 140Ω. R1 was chosen as 142Ω (120Ω + 22Ω).

Using 10% tolerance resistors and assuming worst-case values for these, the maximum and minimum values for the output would be 2.15V and 1.85V respectively. The SA1100's DC specifications required the internal power supply voltage (VDD) to be between 1.90V and 2.10V. Clearly 10% resistors would not be sufficient. If 5% tolerance resistors were used, the output values would be between 1.92V and 2.07V, which is within, but very close to, the limits. Therefore 1% tolerance resistors were used which resulted in the output being between 1.978V and 2.008V. These values allow for temperature and load effects that may change the output.

Using this reasoning, the resistor values for the 3.3V regulator were calculated to be R1 = 240Ω and R2 = 386Ω (377Ω used). With these values the output should be between 3.21V and 3.29V if 1% tolerance resistors are used. This output range falls well between the 3.00V to 3.60V I/O voltage (VDDX) range specified in the SA1100 manual.

	V_{OUT} Minimum (V)	V_{OUT} Maximum (V)
VDD specified range	*1.90*	*2.10*
Using 10% resistors	1.85	2.15
Using 5% resistors	1.92	2.07
Using 1% resistors	1.978	2.008
VDDX specified range	*3.00*	*3.60*
Using 10% resistors	2.93	3.75
Using 5% resistors	3.11	3.51
Using 1% resistors	3.21	3.29

Table 3-1 Summary of voltages using different tolerance resistors

The capacitor values used were 0.1µF, as shown in the circuit in Figure 3-3. This capacitor is used mainly to short-circuit any high frequency pulses, which may occur on the power supply line, to ground. The other capacitors chosen were 10µF instead of 1µF, to yield better stability for the input voltage and the output voltage to the rest of the PCB.

The L7805CV fixed 5V regulator circuit is much simpler and is shown in Figure 3-4.

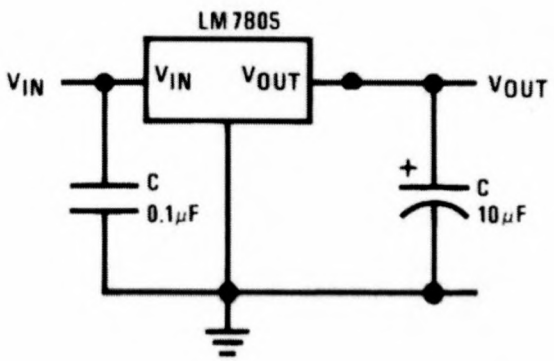


Figure 3-4 L7805CV Power regulator circuit

On this board a 10µF capacitor was added at the input side to help keep the input voltage constant. In this configuration the manufacturer's datasheet specifies an output between 4.8V and 5.2V, which is within the 4.75V to 5.25V specified ranges for the Flash memory, SRAM, EPROM and level shifters.

To protect the board against being connected to an external power supply with the wrong polarization, a 1N4007 diode was put in series with the +VCC power supply. If the power were to be connected up the wrong way, the diode would be in reverse biased state and would not allow any current to flow to the board.

The total power consumption of the board is not expected to be more than 1-2Watts, distributed over the three regulators with the L7805CV carrying the biggest load. This low consumption means that, for now, at least, no heat sinks are necessary on any of the regulators.

3.3.3 RS232 serial communication design

To be able to communicate effectively with other devices and systems, e.g. a Personal Computer (PC), it was necessary to put a RS232 line driver on the board. The schematics for these devices are shown in Figure 3-5.

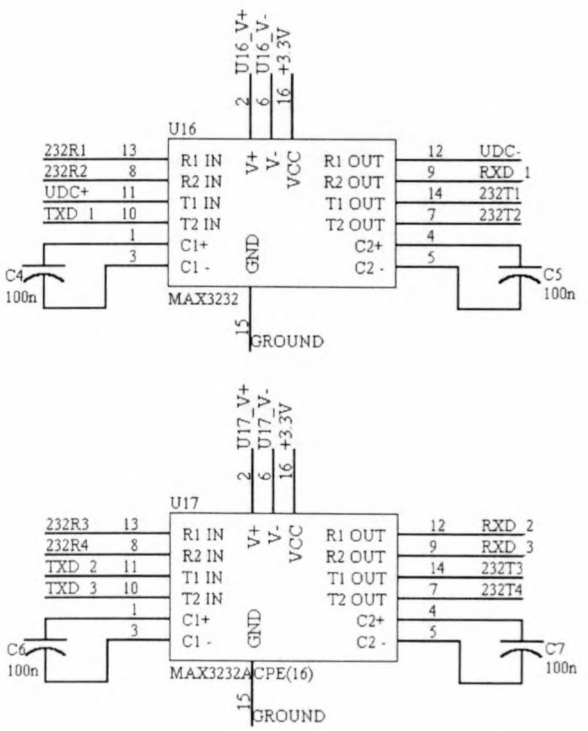


Figure 3-5 RS232 circuit using MAX3232 line drivers

The specific driver used is the MAX3232 dual port driver. The reason this device was chosen is because it uses a 3.3V supply voltage; and can thus be connected directly to the SA1100 processor. The MAX3232 furthermore is very easy to use, needing only four capacitors connected to it to function fully. It is also inexpensive and readily

available. The SA1100, however, has four 2-line serial communication ports, and therefore two tranceiver ICs were needed. With all four serial ports available it is possible to connect to four sub-systems, or other external systems, simultaneously. Figure 3-5 shows the circuit and configuration used for these two IC's.

3.3.4 Memory bus design

On this evaluation board the central bus architecture, as described in 3.2, was used. With this architecture it was necessary to make sure that all the components ports connected to the busses could be tri-stated, so that it would not interfere with the working of the bus while the component was not in use. The EPROM, Flash and SRAM all have the functionality of putting their port pins in tri-state when the devices are not selected. A more detailed block diagram of the memory structure is shown in Figure 3-6.

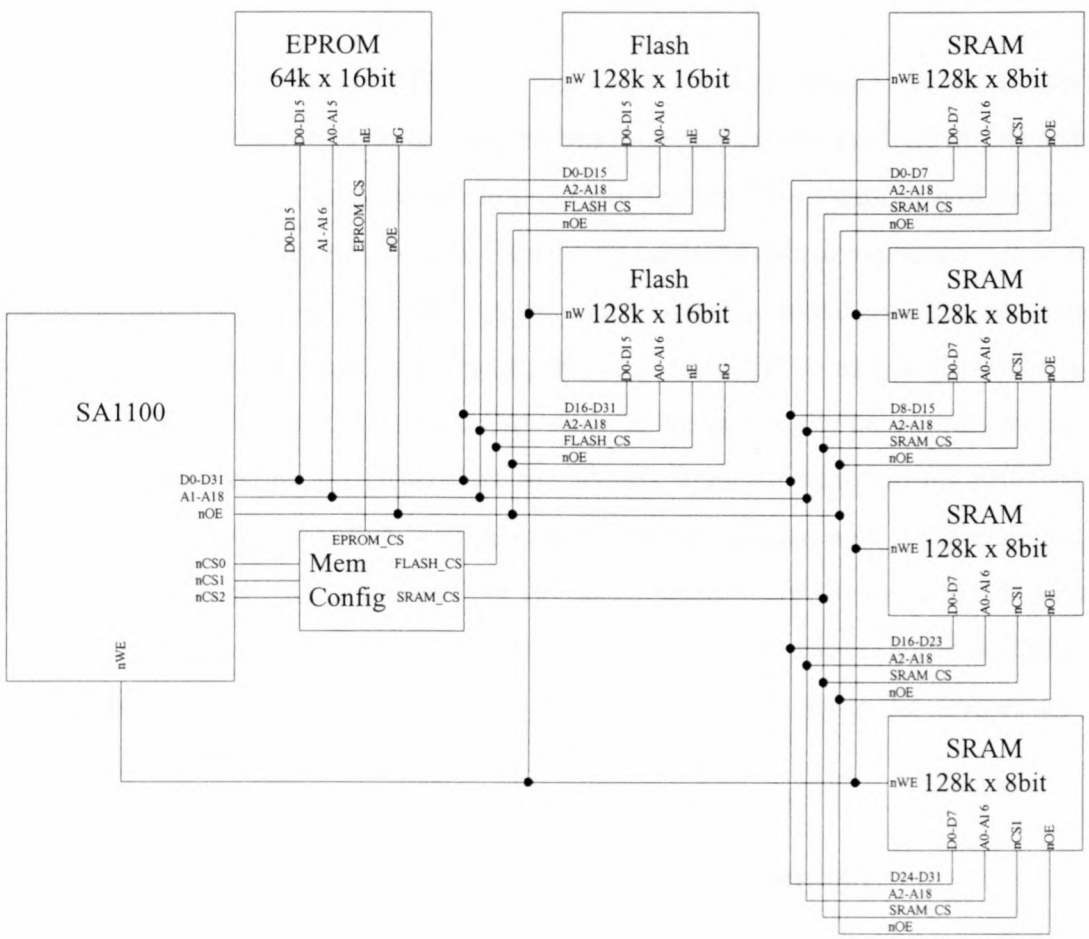


Figure 3-6 Memory system block diagram

The system is configured for the processor to have a 16-bit data bus to the EPROM memory. The code present in this memory will be boot-up code, which is read directly after a hardware or software reset. Ideally, this device should be a Fusible Link PROM (FLPROM) in the final design of a system working in the space environment. Because the boot code must still be developed, it is necessary to have a re-programmable device which can be programmed a large number of times. The EPROM chip-select pin is controlled by the SA1100's CS0 signal. The Flash memory and SRAM are also controlled by the SA1100, using the CS1 and CS2 signals. The output-enable pins of all the onboard memory is controlled by using the nOE signal from the SA1100. The nWE signal controls the write-enable pins for the SRAM and Flash devices. The Flash RAM and SRAM are configured to form a 32-bit data bus, using the four (8-bit wide) SRAM devices and the two (16-bit wide) Flash devices. The data bus is thus split up in D0-D15 for the first Flash device and D16-D31 for the second Flash device. For the SRAM devices, the data bus is split into D0-D7, D8-D15, D16-D23 and D24-D31 respectively.

Viewing the SA1100 from the EPROM requires the address lines to be shifted. When the SA1100 puts an address on the address bus it corresponds to the byte (8-bit) address of the data. This means that, when the processor fetches 32-bit data, it will have to do two accesses to the EPROM (16-bit). Every time EPROM data is requested, a half-word aligned (16-bit) address will be put on the address bus. To make sure that no memory space is wasted, the address line mapping between the SA1100 and the EPROM is not one-to-one. The address that the EPROM wants to see is actually the address put out by the SA1100 divided by two. Therefore the address lines connected to the EPROM are not A0-A15 but rather A1-A16. The same principle was applied with the mapping of the SRAM and Flash memory. The address put on the address bus, by the SA1100, is word (32-bit) aligned, and the address for the devices needs to be divided by four. Therefore the address lines were shifted two places and the lines connected to the devices are A2-A18 instead of A0-A16.

The memory configuration section will be described in Section 3.4.1.

3.3.5 Level shifter circuits

To use the 5V memory devices with the SA1100 processor, which only has 3.3V I/O ports, requires a type of voltage conversion for the address bus, data bus and memory control lines. For this purpose, it was decided to use logic level shifters. These logic

devices have dual 8-bit, 16-bit or 32-bit wide ports. Each port needs its own power supply line.

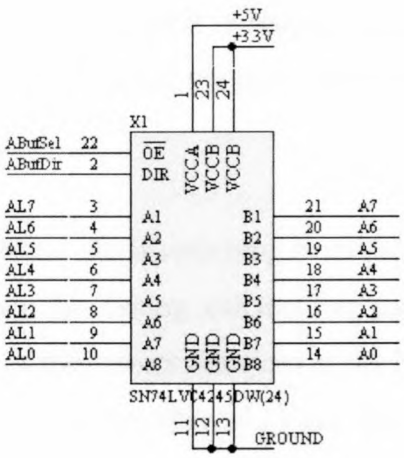


Figure 3-7 Voltage level shifter circuit

Figure 3-7 shows the circuit used. The AL port is the low voltage port for connection to the 3.3V SA1100, and port A is the 5V port connecting to the memory devices. The level shifters are bi-directional, with the direction being determined by the voltage level on the DIR pin of the device. For the address lines and the memory control signals, the direction of the level shifters need never change, because the SA1100 will never receive inputs from these lines. On the data bus, however, it is necessary that the direction of the level shifters be changed according to whether data is read or written to the memory from the SA1100. Therefore it was necessary to use some signal to drive the DIR pin on the level shifters. There are two possible signals that can be used for this purpose, since both can be used to differentiate between write and read cycles. The two possible signals are nWE and nOE. For this design, the nOE signal was used.

To use this signal for directing the data bus, the timing diagrams have to be compared. There has to be enough time between when the nOE signal goes low and when the data must be available to the SA1100. Looking at the 74LVXC4245 datasheet, the switchover time from one direction to another direction is no more than 16ns. With the SA1100 running at its maximum of 221.2MHz, this time translates to four CPU clock cycles. When changing from reading to writing direction, the time that the data bus takes to go into high-Z after the nOE signal goes high depends on the memory used. For the SRAM, the time needed is 25ns, for the EPROM it is 30ns and for the Flash memory it is 20ns (see Appendix D). All three timing maximums exceed the time it

takes the level shifters to change. When the nOE signal goes low, i.e. changes from no data on the data bus to memory putting requested data on the bus, the required times taken by the memory to output the data also depends on the memory used. For the SRAM, this time is 35ns, for the EPROM it is 50ns and for the Flash memory it is 35ns (see Appendix D). All these times exceed the time taken by the level shifters to change direction.

The delay the level shifters cause when switching from the data bus from one direction to the other does not affect the timing calculations used when programming the SA1100. Further, accessing of the memory devices at the highest possible speed, using the nOE signal from the SA1100, is not affected by the level shifters' time delay.

3.4 Configuration designs

The configurations referred to here are mostly preset values or signals needed by the SA1100 processor at start-up.

3.4.1 Read Only Memory (ROM) width configuration

An interesting feature added to the design was the ability to switch between booting from EPROM and booting from Flash memory. To achieve this, a set of jumpers was added to the board. The SA1100 starts executing code from address 0000 0000H. This address corresponds to the CS0 signal, as shown in the memory map in Figure 2-5. The jumpers can be set to select the memory devices connected to the CS0 line, thus either the Flash memory or EPROM can be used to boot from. Figure 3-8 shows the jumper connection diagram.

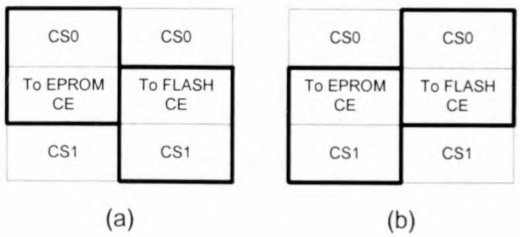


Figure 3-8 Memory select jumpers to boot from (a) EPROM or (b) Flash

In addition to the above settings that are needed to change the boot memory device, there is also another jumper that needs to be set, the ROM_SEL signal jumper. The

SA1100 uses this signal to determine the bus width of the boot memory. For the EPROM, which has a 16-bits wide data bus, this jumper must be set to ground, and for the Flash memory, which has a 32-bit wide data bus, it must be set to VCC. The ROM_SEL signal is checked by the SA1100 at start-up and is only used for the boot memory.

3.4.2 Test clock mode

The SA1100 has the ability to be driven by an external clock source. This is done by connecting the TESTCLK pin to the external clock source and setting up the TCK_BYP pin. If the TCK_BYP pin is driven high, the TESTCLK signal is used as core clock instead of the internal PLL. When TCK_BYP is driven low, the internal PLL is used as core clock. This feature is very useful when slow speed debugging is required, since the program can be stepped through at leisure by just toggling the TESTCLK signal.

3.4.3 External Power Supply Interface

As explained in chapter 3, low power consumption is a very useful and desired quality of this processor. In order to utilise this feature, an intelligent power supply has to be used with this processor. This power supply must be able to communicate with the processor. With the data received it must be able to switch the supply to the processor on or off. Three signals are used for this purpose. The first is the PWR_EN signal which the SA1100 uses to signal the external power supply that it is going into *sleep* mode and that the VDD power supply should be removed to conserve power. This signal is active high. The other two signals are signals that the external power supply sends to the SA1100. VDD_FAULT tells the SA1100 that the VDD power supply is going out of regulation. This causes the SA1100 to go into *sleep* mode and VDD_FAULT is ignored from when the processor wakes up until the power supply timer completes (approx. 10ms). The BATT_FAULT signal tells the SA1100 that the main power supply is being interrupted. When this signal is received, the SA1100 enters *sleep* mode and will not recognise a wake-up event while the BATT_FAULT signal is asserted.

On the evaluation board, the PWR_EN signal is connected to a header, which can be used when an intelligent power supply is connected to the board. The BATT_FAULT and VDD_FAULT signals are connected to jumpers that set them to the default "on" mode or connect them to the external power supply.

3.4.4 Reset source

Implemented on the evaluation board is a reset circuit that is used when power is first applied. This is a simple circuit using the charging curve of a capacitor through a resistor. The RC time constant is chosen to be big enough to ensure that no more transients from the power supply are present. This circuit is also used to provide a hardware reset source. Figure 3-9 shows the reset circuit implemented.

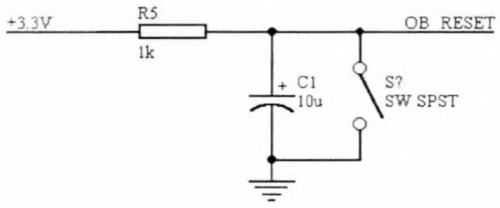


Figure 3-9 SA1100EVB reset circuit

By pressing the switch, the nRESET signal is negated. The circuit then takes time, proportional to the time constant of the RC circuit, to charge the capacitor, ensuring the SA1100 resets safely. This time constant can be calculated as the product of the resistor value and the capacitor value:

$$\begin{aligned}\tau &= R \times C \\ &= 1 \times 10^{3} \times 10 \times 10^{-6} \\ &= 10ms\end{aligned}$$

3.4.5 Address and Data bus direction and selection

The 74LVXC4245 device has an enable pin and a direction pin to control data flow through the device. The enable pin is used to switch the device on or, if negated, to disconnect the two ports on the device by switching it off. The direction pin on the device is used to control the direction of data flow. The direction and enable pins for the devices located on the address bus can be set via jumpers to default values (i.e. direction is always away from SA1100 and the devices are always enabled), or to be controlled by an external bus controller. The devices connected to the data bus can be set via jumpers, to be enabled or to be controlled by an external controller. The direction pin of the devices connected to the address bus can, however, not be changed to any source other than the SA1100's nOE signal.

3.4.6 Debugging Sections

When designing new boards, a large amount of time will always be spent on debugging the hardware. This was kept in mind when the SA1100EVB was designed. To help with debugging, all the pins from the SA1100 that are used for configuration and control were connected to headers on the EVB. This makes it easy to access the pins. Other circuits built into the SA1100 board for debugging purposes were the test LED array, test DIP-switch array and the power indicator LEDs. Usually only one LED is used to show whether the power to a board is switched on. In this design, each power supply on the board has its own indicator LED, i.e. the 5V supply, 2V supply and the 3.3V supply. The LED array and DIP-switch array can both be used with the help of a ribbon cable, for debugging up to 8 lines at the same time.

4 Software development

A large amount of time was spent on developing software for SUNSAT, as indicated in Section 2.5. Most of the software that was written should ideally be reusable for future satellite programs. Two major factors have to be considered when software needs to be reusable. The first is the programming language being ported to. The second is the way the software is written.

For this study it was important to write software to test the SA1100EVB. The first part of the software had to be written in Advanced RISC Machines (ARM) assembler. The second part was written in the ANSI-C programming language. Before the programming could be done, however, the programming environment needed to be set up.

4.1 Programming environment

The SA1100 runs on assembler code that is compatible with the ARM V4 architecture instruction set [6]. To generate code for the SA1100, an inexpensive compiler package was needed. A number of very good programming suites are available. One of these is the ARM Software Development Kit (SDK). The problem with this kit is that it is very expensive. It was decided to rather use the ARM-ELF GNU Cross Compiler (GCC) package, which is available with most Linux distributions, with no licensing or other fees. This GCC package is distributed under the GNU public license, which means that its use and distribution is free. One of the problems of creating a programming environment for a project is making the environment accessible to other programmers who want to continue from where this project stops. Some time was spent in the beginning of the programming phase to set up the environment in such a way that anyone in the laboratory could have controlled access to it. The GCC tool chain runs on the Linux operating system. For the environment to be set up correctly, some help was required. H. Grobler (M.Eng) installed the tools on an existing Linux server and set up the necessary permissions. F.G. Retief and A. Barnard (author) did the customisation of the tools. Customising the tools involved changing the configuration files for the cross compiler and linker to use the ARM-ELF compiling tool chain instead of the standard tool chain. The environment can be used by anyone who has an account on the Linux server, with minimal customisation. Before the

processor can be programmed a little more must be said about the ARM V4 architecture and instruction set.

4.2 More about ARM V4

To be able to write any program efficiently, one has to understand the hardware you are working with and how the hardware reacts to the instructions given to it. It was therefore necessary to do a short study of the ARM V4 architecture and assembler code.

ARM V4 instruction set has only 45 instructions. The full instruction set and its description is given in [6] and is too extensive to be listed in this report.

The ARM architecture incorporates these typical RISC architecture features [6]:

- large uniform register file
- load/store architecture, data-processing operations that only operate on register contents, not memory
- simple addressing modes, load/store addresses determined from register contents and instruction fields
- uniform and fixed-length instruction fields, which simplify instruction decoding.

The ARM architecture, in addition, provides:

- control over Arithmetic Logic Unit (ALU) and shifter in data-processing instructions
- auto-increment and auto-decrement addressing modes
- Load/Store Multiple instructions
- conditional execution on all instructions

The ARM has 31 general-purpose 32-bit registers, of which 16 are visible at any one time. It also has 6 status registers of 32-bit width. Registers are arranged in banks as shown in Figure 4-1. The general-purpose registers R0-R15 can be split into three groups:

- Unbanked registers - R0-R7 are the same physical register for all modes
- Banked registers - R8-R14 are different physical registers specific to each mode
- R15 - the Program Counter

The banked registers make it possible to write very efficient code, because the storage of register values is simplified. Reaction time when FIQs occur can be especially fast when effective interrupt handlers are used.

There are some special registers that are described in the following sections.

4.2.1 R13 - Stack Pointer

This register is normally used as stack pointer if the ARM instruction set is used. The ARM Thumb instruction set, however, has some instructions that use R13 for other special functions. Each exception mode has its own banked R13, and therefore can have its own independent stack. The exception handler uses these stacks to save other registers so that it does not corrupt the state of a program when an exception occurs.

Modes						
	Privileged Modes					
		Exception Modes				
User	**System**	**Supervisor**	**Abort**	**Undefined**	**Interrupt**	**Fast Intr**
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

Shows that original register has been replaced by alternative banked register, specific to the exception mode

Figure 4-1 ARM Register organization

4.2.2 R14 - Link Register

The Link Register (LR) has two special functions in the architecture:

- to hold subroutine return addresses that are copied back to the program counter when the subroutine ends
- to hold exception return addresses, when an exception occurs, that are copied back to the program counter when the exception return is performed

At all other times R14 can be treated as a general-purpose register.

4.2.3 R15 - Program Counter

The program counter can be used as a general-purpose register, but only under certain restrictions. It is always used for a special purpose when:

- reading from the program counter - returning the address of the instruction plus 8 bytes, used for quick position-independent addressing
- writing to the program counter (address must be word aligned) - in effect produces a branch to the address equal to the value written

4.3 The programming code structure overview

In order to make the development cycle shorter and the debugging of code faster, the aim of the process is to provide a system that can be upgraded and changed easily. The hardware already gives some aid by providing the possibility of booting from EPROM, as well as from Flash memory. The software must provide the means to utilise this hardware feature. Instead of waiting ten to fifteen minutes every time to erase the EPROM when new code must be tested, the code can be loaded into the Flash memory, via the RS232 port, from the PC. This will enable the developer to test new code on the board quickly and easily. To achieve this, a certain code structure had to be defined.

Two distinct code sections that could be defined. The first was the start-up configuration code, responsible for initialising the SA1100 processor. It enables two-way communication with the PC and optimal access to the memory. This can be called the SA1100 driver software. It is preferable to put some code in assembly and some in C-code. The assembly code is needed to set up the CPU speed, interrupt vectors, UART communication, and to set up the hardware in order to use the C-code. This means that a stack pointer must also be set up in the assembly code. The second code section is the actual application. Ideally this section must provide the functionality to upload code to the memory from the PC and to perform low and higher level debugging hardware and software tests. The application should also have the functionality to load an operating system.

4.4 The assembly boot-up code

The previous section mentioned that a part of the code should be written in assembly. This section of code was developed first, since low-level hardware debugging of the board cannot be done without it. The boot-up assembly code needs to set up a few registers in the SA1100, enabling the system to switch to the C-code. The first of these settings is the interrupt vector table.

4.4.1 Interrupt vector table

After a reset, the SA1100 starts executing code from address 0h0000 0000. The interrupt vector table, therefore, has to be set up at the beginning of the code, since the compiler places this code at the start addresses in the memory. The complete code is listed in Appendix C. The first line of code jumps to the `reset` procedure. This procedure starts configuration of the SA1100.

First, all the interrupts are masked so that none can occur during the set up procedure. The Interrupt Controller Mask Register (ICMR) must be initialised. If a zero is written to any bit in this register, the interrupt corresponding to the bit will be ignored. Figure 4-2 shows the ICMR of the SA1100. The question marks indicate that the value of the bit in the register in unknown after reset, and the register must be initialised to known values after each reset. To mask all interrupts, 0h0000 0000 is written to the ICMR.

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R/W	IM31	IM30	IM29	IM28	IM27	IM26	IM25	IM24	IM23	IM22	IM21	IM20	IM19	IM18	IM17	IM16
Reset	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R/W	IM15	IM14	IM13	IM12	IM11	IM10	IM9	IM8	IM7	IM6	IM5	IM4	IM3	IM2	IM1	IM0
Reset	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Figure 4-2 SA1100 Interrupt Controller Mask Register

4.4.2 Initialising Central Processing Unit (CPU) core clock-speed

To set the CPU to the desired speed, the correct value of the core Clock Configuration Field (CCF) must be written to the Power manager PLL Configuration Register (PPCR). The register is shown in Figure 4-3. Table 4-1 shows the obtained PLL frequency

corresponding to the written CCF value, as given in [4]. In this case the CCF value is set to 0x00000 (corresponding to a core clock frequency of 59.0MHz).

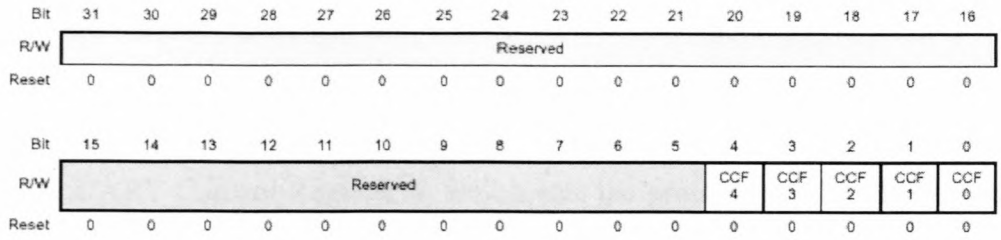


Figure 4-3 SA1100 Power Manager PLL Configuration Register

CCF<4:0>	Core Clock Frequency in MHz
	3.6864MHz Crystal Oscillator
00000	59.0
00001	73.7
00010	88.5
00011	103.2
00100	118.0
00101	132.7
00110	147.5
00111	162.2
01000	176.9
01001	191.7
01010	206.4
01011	221.2
01100-11111	Not supported

Table 4-1 Core Clock Configurations

4.4.3 Universal Asynchronous Receiver / Transmitter (UART) initialisation

Communication from the SA1100 is very helpful during debugging and to determine whether all the settings were done as intended. It is, therefore, necessary to initialise one of the serial UART ports, to transmit the information to the PC. The procedure named `txstart` is given in Appendix C and shows the code to initialise the UART port and to start transmitting some characters to the PC.

The procedure first clears all the "sticky" bits in the UART Status Register 0 (UTSR0). Sticky bits refer to bits, set by hardware, which can only be cleared by writing ones to them via software. Writing zeros to these bits has no effect on their status. The UTSR0 is thus cleared at start up.

The UART control registers must be set to the desired values. Figure 4-4 shows the SA1100's UART Control Register 0, which sets the protocol used by the UART when transmitting and receiving data. The program sets the register's value to 0h0000 0008. This value translates to the following settings:

- PE = 0, Parity checking on received data and parity generation on transmitted data is disabled.
- OES = 0, Odd parity checking/generation selected. Parity error bit set if even number of ones is counted in data field (including the parity bit).
- SBS = 0, One stop bit transmitted per frame.
- DSS = 1, 8-bit data.
- SCE = 0, on-chip baud rate generator and digital PLL used to transmit and receive asynchronous data.
- RCE = 0, Rising edge of clock input on GPIO pin 20 used to latch data from the receive pin if SCE=1.
- TCE = 0, Rising edge of clock input on GPIO pin 20 used to drive data onto the transmit pin if SCE=1.

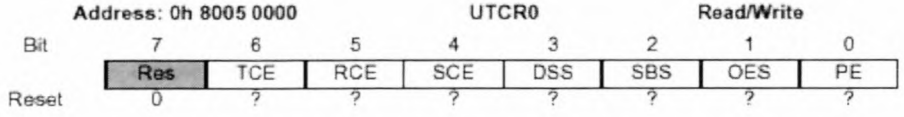


Figure 4-4 UART Control Register 0 (UTCR0)

The settings are not of critical importance and were only chosen to start off with, since most terminal programs on the PC can handle them. The UART Control Register 3 (UTCR3) also needs to be initialised. Figure 4-5 shows the UTCR3, and at initialisation all the bits are set to zero, translating to:

- RXE = 0, UART receive operation disabled; Peripheral Pin Controller (PPC) is given control of **RXD**3.
- TXE = 0, UART transmit operation disabled; PPC is given control of **TXD**3.

- BRK = 0, UART in normal operation.
- RIE = 0, Receive First-in-first-out buffer (FIFO) one- to two-thirds full (or more) and receiver idle conditions do not generate an interrupt (RFS and RID bit ignored).
- TIE = 0, Transmit FIFO half-full (or less) condition does not generate an interrupt (TFS bit ignored).
- LBM = 0, Normal serial port operation enabled.

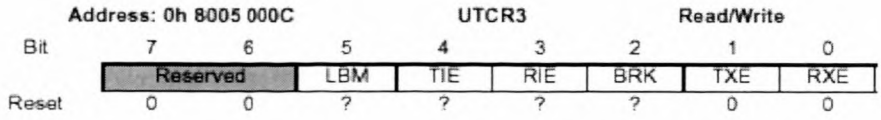


Figure 4-5 UART Control Register 3 (UTCR3)

Both the UTCR0 and UTCR3 registers have reserved bits in them. These reserved bits are used by the SA1100 internally and is not accessible to the programmer. Any values written to these bits are ignored, while reading from these bits returns zeros.

The last initialisation, before the transmitter can be enabled, is to set up the baud rate for the port. This is done by programming the correct Baud Rate Divisor (BRD) field value into the UTCR1 and UTCR2 registers. For the SA1100 UART ports a total of 4096 baud rates can be programmed, ranging from 56.24 bps up to 230.4 bps. The baud rate generator uses the 3.6864MHz clock from the on-chip PLL divided by 16 to generate the bit clock. To calculate the BRD to be programmed when a certain baud rate is needed, the following equation can be used [4]

$$BRD = \frac{3.6864 \times 10^6}{16 \times BAUD\ RATE} - 1$$

A baud rate of 9600 was initially chosen. Using the above equation the BRD value that must be written to UTCR1 and UTCR2 was calculated to be 23. This value is written to the registers by `txstart`.

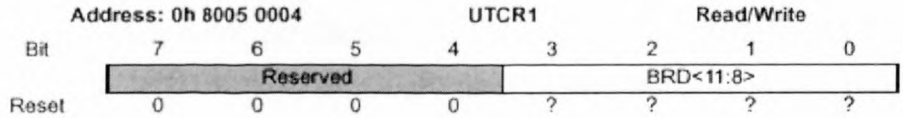


Figure 4-6 UART Control Register 1 (UTCR1)

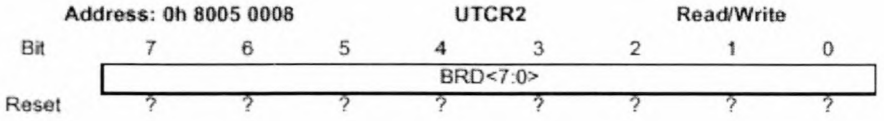


Figure 4-7 UART Control Register 2 (UTCR2)

Finally the transmitter must be enabled, by writing a one to bit number two in the UTCR3.

4.4.4 Checking the SRAM

All the cache memory on the SA1100 will be disabled when used in space, but the processor needs memory for the stack initialisation before the C-code application or Operating System (OS) initialises its own memory management system. The last procedure that must be done before jumping to the C-code is testing the SRAM integrity.

The procedure named `ram_main` was written to test the SRAM. The first part of the procedure initialises the Static Memory Control 1 (MSC1) register. This register is shown in Figure 4-8. Bits 0-15 of the register is used to set the timing for access to the memory controlled by the CS2 signal from the SA1100, which is the SRAM in this case. These bits all have unknown values after reset and, therefore, needs to be initialised before the SRAM can be reliably used.

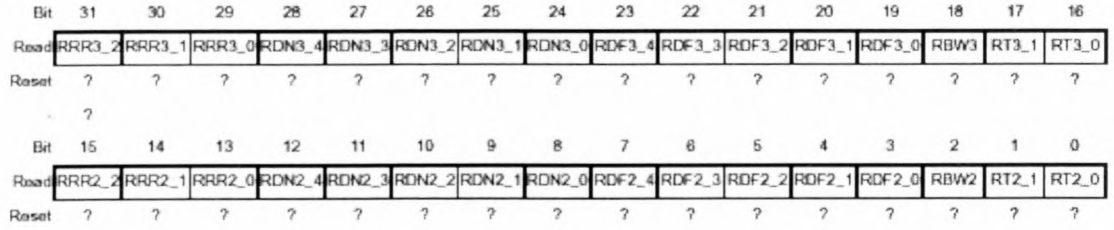


Figure 4-8 Static Memory Control 1 (MSC1) register

There are three "timing" fields that must be set, and two "type" fields to set the memory type. The three timing fields are the ROM Delay First access (RDF), ROM Delay Next access (RDN) and ROM/SRAM Recovery time (RRR) fields. The two type fields are

the ROM Type (RT) and ROM Bus Width (RBW) fields. For the SRAM the RT field must be set to 01 as seen in Table 4-2.

The RBW field must be set to 0 for the SRAM, since the SA1100 can only use a SRAM configuration with a 32-bit wide data bus.

RT Value	Description
00	Nonburst ROM or Flash EPROM.
01	Nonburst ROM or SRAM.
10	Burst-of-four ROM.
11	Burst-of-eight ROM.

Table 4-2 ROM Type (RT) field value definitions

The SRAM was set to the slowest possible times. The RDF and RDN fields are both set to 11111. The RRR field is set to 011, which translates to 6 memory cycles or 12 CPU cycles. This setting gives a recovery time of at least 54ns.

After the initialisation, the procedure writes some data to the SRAM and then reads it back. The data read is checked against the written data and the errors are counted. This test is done for all the SRAM memory addresses. If the test is successful, the procedure continues to the next section of the program.

4.4.5 Stack pointer initialisation

The C-code needs a block of memory assigned to it for a stack. To allocate this memory, the assembly code must pass the size and start address of this block of memory to the C-code in the form of arguments when jumping to it. The SRAM memory space starts at 0h1000 000. For the first tests, the C-code would not need all the SRAM memory and a memory block of 0h0001 0000 was decided on. The first argument passed to the C-code is the start address and must be loaded in R0, the second argument is the size of that block and must be loaded in R1. Before the jump to the C-code can be made, the stack pointer must be moved to the top of the memory block that is available to the C-code. The stack is a top-to-bottom loading structure; therefore the pointer is moved to 0h0FFF FFFC. This value is the top memory position minus 4, because the stack pointer must point to the next address to be used in the stack.

4.5 Possible operating systems

SUNSAT used the RTX operating system written at the ESL. This operating system has since been ported to ANSI-C language from the original Modula2 language. RTX is therefore ready to be used on a system like the SA1100 evaluation board. However, reviewing the software written for the SA1100EVB so far, shows that it does not provide enough hardware support or functionality to enable RTX to run on the system. There are a few memory management procedures that must still be implemented.

Work being done on a Java Virtual Machine (JVM) at the ESL by H. Venter, aims to incorporate most of RTX's functionality. This would enable software developers working on future software for the SA1100EVB and peripherals to write in the native JAVA language. This is a very useful feature, since specialist computer programmers could then be used to write programs in the user-friendlier JAVA. The JVM however is not finished yet and the software libraries that it needs still have to be written.

5 Test and debugging

5.1 Hardware debugging

The hardware debugging started after the PCB was made. The PCB had to be checked to eliminate any of the following defects:

- Short circuits between power supplies and ground
- A short-circuit between any power rail/ground and any pin not supposed to be connected to that power rail/ground
- An open circuit between points that ought to be connected, i.e. checking that there are no breaks in the tracks
- Incorrectness of the silk layer, i.e. checking that component names correspond to the correct places on the PCB
- Incorrectness of connections between Data, address and control lines and their corresponding pins

During hardware debugging a few design flaws and errors were encountered and corrected. The following paragraphs will describe these errors and the actions taken to correct them.

5.1.1 EPROM address line mismatch

An important factor that was not considered when the original hardware design was made was the difference in addresses, as seen by the SA1100 and the different memory devices.

The SA1100 works in byte-sized data, i.e. each address it uses is only for 8-bit data. The EPROM, in comparison, is a 16-bit wide device. Therefore when the SA1100 sends out an address to read from it, it is an absolute address (as if 8-bit devices were directly connected to the address bus). Because the EPROM has a 16-bit wide data bus, for each address it receives, it outputs 16-bit wide data. The addresses sent out by the SA1100 to the EPROM, therefore should be divided by two (in hardware) for the EPROM to be fully utilised. This is easily done by shifting the address lines connected to the EPROM by one line and effectively dividing by two. Instead of the EPROM receiving A0-A15, it now receives A1-A16 on its address port. Putting in an additional EPROM socket on

top of the original socket and changing the connections between the two sockets corrected the error. The additional address line needed was connected to the socket with a wire from the address bus header.

The same type of problem was encountered with both the SRAM and the Flash memory. For these two memory blocks the address lines needed to shift two places, because the devices are 32-bit wide data instead of 8-bit wide data. The Flash memory, as well as the SRAM, are surface-mounted devices and changing connections proved more of a challenge than with the EPROM. Addresses from the SA1100 must be divided by four to get the correct mapping to the memory. Breaking the A0 and A1 tracks connected to the block and connecting the A17 and A18 lines from the address bus header to the SRAM block instead, corrected the SRAM connection problem. With the Flash memory it wasn't as easy. Since the Flash memory needs certain commands before data can be written to it, it is critical that one-to-one mapping, together with a two-bit shift, is implemented. This meant that each address line connected to the Flash memory had to be shifted. The surface-mounted ICs' pins that were connected correctly were left connected to the PCB, while the other pins were straightened and lifted from the PCB. Wires were connected between the lifted pins and the correct pads on the PCB, to correct the mismatch.

5.1.2 SRAM Chip Select not connected

The hardware design was done on Design Explorer 99 SE (product of Protel International Limited), which is the standard computer-aided design tool currently used in the ESL. Due to a tight schedule the design was started before all the required components were ordered. It was decided to use 8-bit wide SRAM because of its earlier use. The manufacturer, however, was not specified. The computer design was done using a generic 8-bit SRAM IC, because libraries for the ordered component were not available. This model was then used through out the whole design. The IC used, however, was not the same as the model. The model's pin 32 was marked as a "Not Connect" pin. The part ordered and used in the final design used its pin number 32 as a second chip select pin. This error was only discovered after the PCB was made and most of the components were soldered onto the board. The not-connect pins (pin number 32) on the SRAM chips were connected to a fixed "high" voltage that ensured that the second chip select line was always enabled. This corrected the problem and the SRAM worked flawlessly.

5.1.3 Absence of a push button reset switch

In the original design, provision was made for connecting an external reset source to the SA1100 that also used the charge-up circuit on the board. However, no switch was included on the board itself to enable the developer to reset the board without disconnecting the power supply. To correct this problem, a push button switch was included and connected to the reset circuit.

5.2 Software debugging

Software debugging is more of a continuous process than the hardware debugging process and it would not be practical to list all the errors encountered while writing the first part of the software. An explanation of a typical software debugging cycle (during this project) can however be given.

5.2.1 The software debugging cycle

The cycle starts with the program being planned as described in section 4.3, then written in a text editor. This editor must preferably have syntax highlighting to assist the programmer in writing the program using correct syntax. This is usually the first place to look for errors in a program.

The next phase is to get the compiler to compile the code that was written. One of two methods can be followed in this phase. While the compiler environment was being set up and tested, the types of programs written were such that they could give output to the PC screen. The compiler used in these tests was the standard GCC compiler. This compiler compiled the programs for use on the PC hardware. The tests could only be executed on code written in ANSI-C, because the standard GCC compiler cannot interpret the ARM assembly. After the code was tested, using the standard GCC compiler, the same code was compiled using the ARM-ELF GCC compiler. Using short C-Code programs, the assembly generated by the ARM-ELF compiler could be compared to the expected code. By using these methods, the ARM-ELF environment can be configured correctly. ARM assembly programs can now also be compiled correctly as a result of these tests.

After a section of the program is compiled, it should be tested. There are several different ways to test a program. The first (most obvious) way is to load the program

onto the hardware and execute it. With this test, it is easy to see if the program works correctly. If it fails to work, however, it can be very difficult to find the error (even with debugging code included in the program). To solve this problem, it is better to first execute this program in a simulated environment, as provided by the GNU DeBugger (GDB) tool. The program can be tested by loading it into the debugging tool and then stepping through it one code line at a time. The code lines can be changed to either be the original C-code lines or the compiled assembler code lines. The memory space of the program and the data are continually updated and in this way programs can be fully debugged. There are some drawbacks to using this method. If the program being debugged becomes large or has a number of repeating loops, it can become quite a task to step through the program line by line. The program should be debugged section by section and the programming ideas should first be tested, before the final program is tested as a whole.

After the program was debugged using software it still had to be tested on the hardware. If the program continues to have problems functioning correctly, some extra debugging code inside the program may be needed. This enables the developer to identify problems, through determining under which circumstances exactly the program caused errors. It also helps to find tricky hardware bugs, like timing problems, that are less easy to identify than a loose connection.

5.3 Functional tests and results

In order to test the ease of use and the performance of the SA1100, a few functional tests where conducted. These tests are very low level tests and are designed to test the integrity of the SA1100's peripherals and to gather more detailed information about the performance of the processor.

5.3.1 The UART performance test

This test is conducted to measure the quality and ease of use of the UART peripherals on the SA1100 the processor. The communication ports 1, 2 and 3 can be configured to function in their respective UART modes. Port 1 can be tested using the normal UART mode, where the RXE and TXE pins are used as the receive and transmit pins respectively, as well as in the GPIO mode, where the GPIO pins 14 (transmit) and 15 (receive) are used as receive and transmit pins respectively.

For this test, serial port 3 was used and its control registers were configured to cover all the possible UART configuration settings. A typical measured trace of a UART transmitted frame is shown in Figure 5-1.

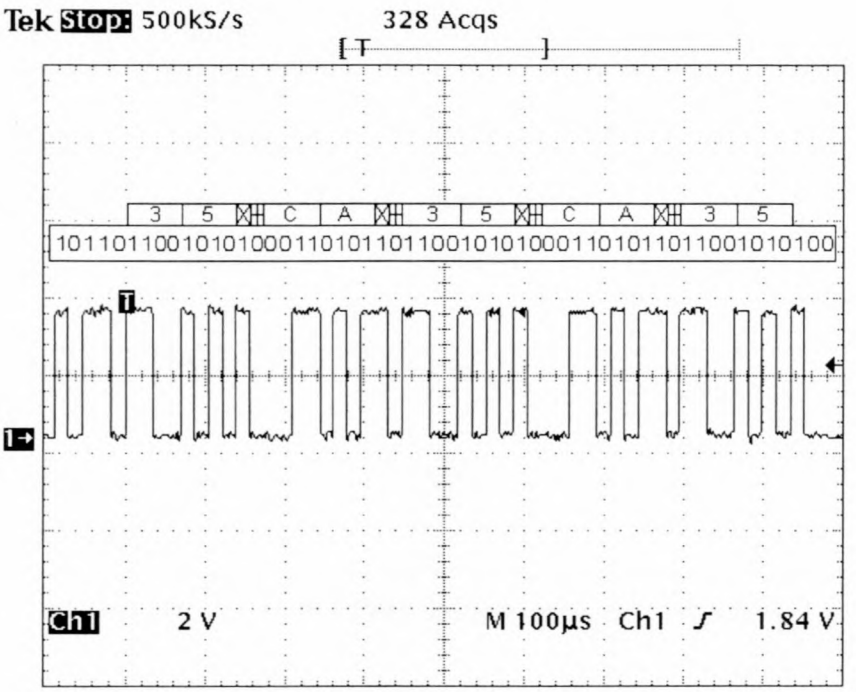


Figure 5-1 Trace of typical UART frame

The data transmitted by the UART was a string of repeated bytes, testing the eight bits per frame configuration. Figure 5-1 also shows the decoded bit values superimposed on the oscilloscope trace. The UART's frame data is transmitted Least Significant Bit (LSB) first and the Most Significant Bit (MSB) last. The hexadecimal values can easily be decoded. On the figure, the start bit is marked by with a "+" sign and the stop bit by an "x" sign.

Table 5-1 shows the measured values of port 1 transmitting data at various speeds. The values measured are accurate within the percentage shown in the maximum possible measurement error row. The reason for the possible variance is that the measurements were taken using a digital oscilloscope, which has a limited resolution (50 pixels per division).

To test the data integrity, the RS232 port of the evaluation board is connected to the PC. On the PC a terminal program is used to receive the data. The received data is then compared to the data expected. The encoding used by the UART, however, is different

from the encoding specified in [4]. Instead of the Non-Return-to-Zero-1's-Invert (NRZ-I) encoding expected, the UART uses Non Return to Zero Level (NRZ-L) encoding.

Set baud rate	**1200**	**9600**	**38400**	**57600**
Measured rate	1199	9615	38314	57636
Error	0,083%	0.156%	0.224%	0.062%
Max possible measurement error	0.239% (2µs over a 833.3µs period)	0.24% (0.5µs over a 208µs period)	0.38% (0.1µs over a 26.1µs period)	0.115% (0.02µs over a 17.35µs period)
Data received correctly?	Yes	Yes	Yes	Yes
Result within expected limits?	Yes	Yes	Yes	Yes

Table 5-1 UART Ports performance measurements

From the results obtained from this test, the UART seems to operate correctly (using NRZ-L encoding) at all the tested baud rates. Because the UARTs of port 1 and port 2 operate the same way as the UART of port 3, this test was not repeated for them.

5.3.2 Real Time Clock (RTC) Test

The SA1100 has a real time clock unit. This unit can be programmed to generate an accurate 1Hz signal. The SA1100 Developers Manual [4] states that the RTC can be calibrated (trimmed) to be accurate to within +/-5 seconds per month. To test this claim, the RTC trim procedure has to be performed. The RTC has a trim register that has to be programmed with the correct value, to generate the 1Hz clock. To be able to program this register correctly the exact frequency of the 32.768KHz oscillator must be measured. The 32.768KHz oscillator has no output that can be measured directly; instead the GPIO pin 27 must be switched to its alternate function configuration. The measurements showed that the frequency is 32710 +/- 268.5 cycles per second (due to the digital scope error). This difference translates to a 0.8% error compared to the trim register that can be set up to an accuracy of 0,000977% (1/1023th of a second). The measured error, therefore, can be up to 80 times larger than the trim error. Unfortunately

the measurements obtained from this test are not accurate enough to continue with the trim procedure.

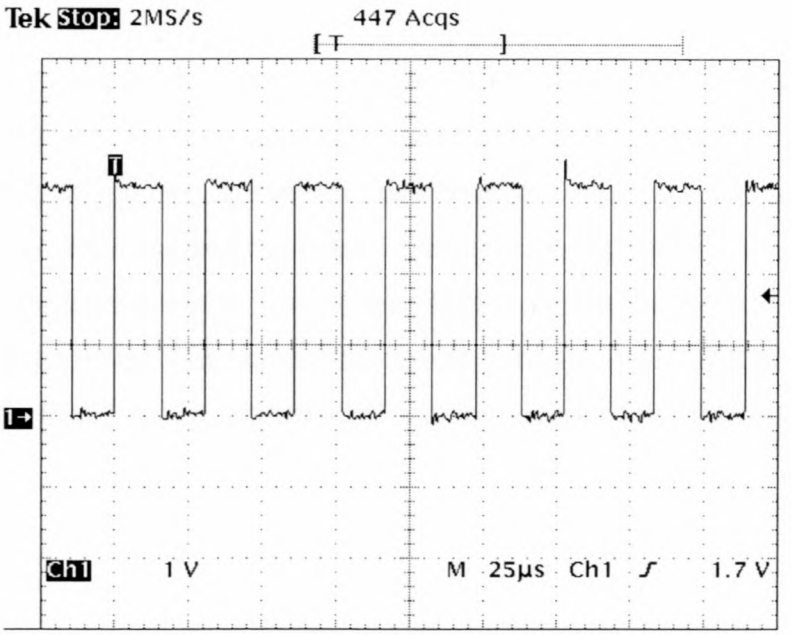


Figure 5-2 32.768kHz Clock on GPIO pin 27

5.4 Power consumption measurements

The amount of power consumed by any system on a satellite is of high importance. It was necessary to measure the power consumption of the processor and the evaluation board. The processor's power consumption could not be measured directly, since no provision was made for such measurements in the design of the evaluation board. To get a good estimate of the power consumption, the processor was put into sleep mode and the current drawn by the board was measured.

In this mode the maximum power that the processor consumes is assumed to be no more than 165µW, as stated in the datasheets. The amount of power consumed by the board at this time is not less than the total power consumption measured minus the maximum amount of power used by the processor (in *sleep* mode). The current supplied to the board was measured to be about 70mA. This means that, if we assume the current drawn by the SA1100 to be no more than 50µA, the board draws about 70mA. The maximum power used by the board is calculated to be 630mW.

To measure the amount of power that the processor uses in *idle* mode is only possible if the processor goes into *idle* mode and leaves all the pins in the same state as when it goes into

sleep mode. Because a good estimate of the maximum amount of power used by the board is available, a good estimate of the amount of power consumed by the processor in *idle* mode can be obtained.

There are, however, some problems with this test. When the processor was put into sleep mode the current drawn by the board increased to 113mA. When the processor was brought out of sleep mode the current dropped back to 70mA. This increase in power consumption was not expected and the reason for this incorrect result must be found. One possible reason for this increase in power consumption is that the SA1100 drives a LED when going into sleep mode. The amount of current drawn by this LED, however, is only 4.9mA. This leaves 65mA that is unexplained and the source of this increase must still be found.

6 Conclusions and recommendations

6.1 Conclusions

From the measurements taken and the results obtained from the tests done on the evaluation board, conclusions can be drawn about the feasibility of using the SA1100 processor in an OBC. However, to be meaningful and useful for future designs of OBCs for use in LEO satellites, these conclusions must be measured against the requirement guidelines set in Chapter 2. The effect that these results can have on future designs must also be addressed.

6.1.1 Architecture

Having a 32-bit architecture complies fully with the set guidelines. There are three disadvantages concerning the design of a system with a 32-bit architecture, when compared to a 16-bit architecture. The first is that the layout of the PCB is more complex, needs much more planning and takes much longer to do. Having 32-bit data and address busses means that the PCB must either be bigger or have more layers to create enough space for all the tracks. The second disadvantage is that a 32-bit interface is needed for the 8-bit SRAM and Flash memory. This means the memory ICs take up more PCB space because four SRAM ICs are needed instead of two, and two Flash memory ICs instead of one. This disadvantage can be overcome by using 32-bit wide SRAM and Flash memory devices, given that they are readily available and not too expensive. The last disadvantage when using a 32-bit architecture is that the EDAC unit must be larger to handle the wider busses. It is possible that the EDAC will also have to be more complex.

The advantage of having a 32-bit architecture is that the rate of data transfer is dramatically increased. This width means that data is transferred twice as fast compared to a 16-bit architecture, and four times as fast compared to an 8-bit architecture. For an OBC using a 32-bit architecture it would be easier to handle larger data transfers and for the processor to do faster calculations with large numbers.

6.1.2 Commercial grade processor

Being a commercial grade processor, the SA1100 inherently has the disadvantage of not being very suitable for high radiation environments. Until radiation tests are performed on this processor it will be very difficult to predict how severe radiation will affect it. It must be assumed that some radiation effects will occur. Any design using this processor must include some protection against possible data corruption and processor failure.

The advantages of using a commercial grade processor are that:

- it is inexpensive, < R400.00 per unit
- it is available, with less than 4 weeks delivery time
- it has good software support, as it is widely used in other applications

Most of the disadvantages of not being radiation hardened can be overcome through proper design for possible data corruption and device failure. The cost of adding these features to the system will be far less than the cost of a radiation hardened 32-bit processor.

6.1.3 Manufacturing process size

This requirement is closely linked to how radiation affects the processor and has the same disadvantages as those described in the Section 6.1.2. The processor is manufactured with the use of a 0.35µm process, which is small compared to the 1µm process used to manufacture the 80188. However, few integrated circuits using processes larger that 0.35µm are still being manufactured. It is therefore the best possible choice with regards to the manufacturing process size.

6.1.4 On-chip cache memory

Having on-chip cache memory is no disadvantage, in the case of the SA1100, since it is disabled by default after reset. No configuration, therefore, needs to be done by software to disable the cache. The speed of code execution thus depends mainly on the access speed of the memory, as is the case with the 80386 system on SUNSAT. The SA1100 system therefore also has no advantage over the previous systems on SUNSAT, regarding this aspect.

6.1.5 Power supply voltage

The processor should preferably have a 5V supply voltage. The SA1100 needs a 2.0V core supply voltage and a 3.3V I/O supply voltage. These low voltage supplies means that single high-energy particles can easier induce an error charge into the silicon or cause a large enough glitch on a line to corrupt data transfer. Unfortunately most new generation processors use 3.3V (or lower) supply voltages.

There is a major advantage to having lower voltage supplies. When the line voltage transitions is so small the power needed to drive the lines are greatly reduced. The SA1100 uses less than half the total power the old 80386 processor used. This makes the SA1100 system very power efficient.

6.1.6 Speed performance

The guidelines specified at least 5 MIPS while the processor is running at 20MHz. Running at its lowest possible speed of 59.0MHz, the SA1100 should deliver at least 66 MIPS. This value corresponds to about 22 MIPS if the processor were to run at 20MHz. Considering the performance; the SA1100 clearly has an advantage over the processors used previously.

The disadvantage of this processor's operating speed is that cannot run at a lower speed than 59.0MHz. This reduces the quarter wavelength of the clock frequency from 3.75m to 1.25m. At the maximum processor speed of 221.4MHz, the wavelength is only 33.8cm. If the processor has to be used at these high speeds, special care should be taken with the board layout to reduce Electro Magnetic Interference (EMI) and possible "cross talk" on tracks.

6.1.7 EDAC Interface

This processor has only one disadvantage with regards to interfacing with an EDAC unit. The disadvantage is that the interface must consist of the full 32-bit address bus and the full 32-bit data bus. This can cause the EDAC unit to use up a lot of PCB space and power. The EDAC must also do error detection on 32-bit wide data, which possibly makes it more complex. Because no direct EDAC support is available on the SA1100, the EDAC can and should be designed as a separate unit. This has advantages of making the whole design more modular, and disadvantages of adding complexity to the timing calculations and functionality of the overall design.

6.1.8 Software support

The guidelines state that the processor should have good software support. A complete software development tool set in the form of the GNU tools is available. This software is available free of charge, with no licensing requirements. The toolset includes an ANSI-C cross compiler, debugger, linker and library files. It is a great advantage for the SA1100 system to have full support for the ANSI-C language. The 80386 software on SUNSAT is obsolete since there is no support anymore for the MODULA-2 programming language, which was used in the design.

Having open source software available facilitates development since a vast number of other people are using the SA1100 in other embedded systems. Work already accomplished by them can be adapted for use on the evaluation board.

6.1.9 Power supply interface

The SA1100 provides a very useful interface to an intelligent power supply. This enables the system to utilise all the power saving features of the processor, as described in section 2.3.4. The advantage of this interaction is that the processor can be set to *sleep* mode whenever there is no need for it. A timer can then wake it up at a later time to perform scheduled tasks. The *idle* mode can also be utilised to save power while the processor I/O sections are still fully operational.

6.1.10 Evaluation board design

From the design of the evaluation board it was noticeable that the system was not very complex. The board was also very easy to work with. Some improvements can, however, be made.

The first would be to change most of the box headers used to normal dual in-line headers. This will make it easier to take measurements on the board. Another improvement would involve adding two DB9 sockets, in place of the current headers, for the RS232 connections to the PC and other devices. The board should also be modified to enable better current measurements. This would enable the developers to obtain exact power consumption figures and would also help to debug and check the board. Overall, the board was a success.

6.2 Recommendations

6.2.1 Boot loader software

The first recommendation with regard to future work that could be done is that a fully functional boot loader program should be developed. This boot loader must:

- configure the SA1100 so that communications to and from the EVB is possible via one of the serial ports
- load software from the PC into memory and boot from it
- have full debug functionality for code running on the EVB
- be user friendly and interact fully with an available terminal program running on either Microsoft® Windows or Linux

This software is essential for effective development.

6.2.2 Evaluation board expansion

The evaluation board needs to be expanded by designing "daughter" boards to test further functionality. Possible expansion boards can include:

- a serial communication board that expands the serial communication capabilities of the evaluation board to the level of the OBCs on SUNSAT
- a fast data transfer board that can test high data transfer between memory and subsystems or between different subsystems
- an intelligent power supply board that can test the full power saving functionality of the SA1100
- an EDAC board to test EDAC functionality and interfacing between the SA1100 and an EDAC unit
- a large memory board to test interfacing between the SA1100 evaluaion board and external memory

The development of these boards should enable developers to test the full functionality of the SA1100.

6.2.3 Radiation test

Even with all the protection of a dedicated EDAC unit and good design practice, the risk of using the SA1100 in a high radiation environment is still high. Quantitative risk analysis can only be done if radiation test results of the SA1100 can be acquired. This can either be done by doing a radiation test on the SA1100 here at the ESL or by

waiting until radiation tests are done on the processor by some other institution and acquiring the results from them.

6.2.4 Design tools

A large amount of time was wasted during the design of the SA1100 due to the fact that the development team did not know the Design Explorer 99 SE from Protel. This is a very powerful package, but to use it effectively requires that some introduction course be given to students. The course could either be offered at postgraduate level or the student could start using it in projects at undergraduate level.

Bibliography

[1] HOROWITZ, P., AND HILL, W. *The Art of Electronics*, second ed. Cambridge University Press, Cambridge, UK, 1997.

[2] GOOSEN, N. Die ontwerp en evaluering van die sekondêre rekenaar in SUNSAT. Master's thesis, University of Stellenbosch, 1996.

[3] GROBLER, H. Aspects affecting the design of a low earth orbit satellite on-board computer. Master's thesis, University of Stellenbosch, 2000.

[4] INTEL® CORPORATION. Intel® StrongARM® SA-1100 Microprocessor Developer's Manual, August 1999.

[5] GüLZOW, P. *No RISC, No Fun!*, http://www.amsat-dl.org/yahue.html, January 2001.

[6] SEAL, D. *ARM® Architecture Reference Manual*, second ed. Addison-Wesley, Harlow, UK, 2001.

[7] MILLER, L.H., AND QUILICI, A.E. *C Programming Language: An Applied Perspective*, John Wiley & Sons Inc., New York, USA, 1987.

[8] UFFENBECK, J. *The 8086/8088 Family: Design, Programming, and Interfacing*, Prentice-Hall International Inc., Englewood Cliffs New Jersey, USA, 1987.

[9] VAN SOMEREN, A., AND ATACK, C. *The ARM RISC Chip: A Programmer's Guide*, Addison-Wesley, Cambridge, UK, 1993

[10] WAKERLEY, J.F. *Digital Design: Principles and Practices*, second ed. Prentice Hall-Inc., Englewood Cliffs New Jersey, USA, 1994.

[11] ZARGHAM, M.R. *Computer Architecture: Single and Parallel Systems*, Prentice-Hall Inc., Upper Saddle River, New Jersey, USA, 1996.

[12] HWANG, K., AND BRIGGS, F.A. *Computer Architecture and Parallel Processing*, McGraw-Hill Inc., New York, USA, 1984.

[13] NEAMEN, D.A. *Electronic Circuit Analysis and Design*, IRWIN, Chicago, USA, 1996.

[14] JOOSEN, M. *LARTware - Mainboard*, http://www.lart.tudelft.nl/lartware/plint/index.php3, April 2000.

[15] JOOSEN, M. *LARTware - Kitchen sink board*, http://www.lart.tudelft.nl/lartware/ksb/index.php3, 12 October 2000.

[16] AMSAT-UK, *Phase3D*, http://www.uk.amsat.org/phase3d.htm, 10 June 2001

[17] GREEN, C., GüLZOW, P., JOHNSON, L., MEINZER, K., AND MILLER, J. The Experimental IHU-2 Aboard P3D. In *Proceedings of the 16th AMSAT Space Symposium*, 1998

[18] SURREY SATELLITE TECHNOLOGY LTD., Surrey Nanosatellite Applications Platform. Technical Publication SSTL-3001-02, 14 June 2000

[19] INTEL® CORPORATION. StrongARM® SA-1100 Microprocessor Evaluation Platform User's Guide, October 1998

[20] BAYKO, J. *Great Microprocessors of the Past and Present (V 12.1.2)*, http://www3.sk.sympatico.ca/jbayko/cpu.html, June 2001.

Appendix A

Processor Tables

This appendix lists the processors found through the literature study. The processors are listed in tables and grouped together in families. These processors do not constitute all the possible processors on the embedded market, but mainly those that could be regarded as a possible option with the processor specifications kept in mind.

1. ARM Family

CPU Name	Manufact.	V_{CC}(V)	$V_{I/O}$(V)	Process(µm)	Temp(C)	MIPS @ MHz
ARM60	MITEL	5	5	1	-40 / +85	21 @ 30
ARM610	MITEL	5	5	1	-10 / +70	25 @ 33
ARM7100	ARM	2.7--5.5	2.7--5.5	NA	0 / +70	18.4 @
ARM710T	ARM	2.7--3.6	2.7--3.6	NA	-40 / +85	NA
ARM7TDMI	ARM	2.7--3.6	2.7--3.6	NA	-40 / +85	NA
ARM720T	ARM	2.7--3.6	2.7--3.6	NA	-40 / +85	NA
ARM740T	ARM	2.7--3.6	2.7--3.6	NA	-40 / +85	NA
ARM7TDMI	ATMEL	3.3	3.3	0.5/0.35	NA	NA
AT75C310	ATMEL	3.3	3.3	0.5/0.35	NA	NA
CW001007	LSILOGIC	1.8/2.5	1.8/2.5	0.25	0 / +115	@50/@80
CW001004	LSILOGIC	3.3	3.3	0.25	-40 / +85	22@25/13@15
BUTTERFLY	MITEL	5/3	5/3	0.7	0 / +70	NA
M-CORE	MOTOROLA	NA	NA	0.36	NA	NA
SA1100	INTEL	1.5/2	3.3	0.35	0 / +70	180@160/250@220

Table A-1 ARM Family Processors

ARM Family (Continued)

CPU Name	Bus (Addr/Data)	UART	Power	Packaging	Cache(I/D)
ARM60	32/32	None	1.5mA/MHz@5V	100P QFP	None
ARM610	32(26)/32(26)	None	NA	144P TQFP	4kB
ARM7100	32/32	1 Full Duplex	70mW@18.4Mhz/3.3V	156P(SM)	8kB
ARM710T	32/32	None	NA	NA	8kB
ARM7TDMI	32/32	None	NA	NA	None
ARM720T	32/32	None	NA	NA	8kB
ARM740T	32/32	None	NA	NA	4/8kB
ARM7TDMI	32/32	None	NA	NA	None
AT75C310	32/32	2 USART	NA	160P PQFP	None
CW001007	32/32	None	NA	NA	None
CW001004	32/32	None	NA	NA	None
BUTTERFLY	22/32	2	NA	144QFP	None
M-CORE	32/32	None	NA	NA	None
SA1100	32/32	4	< 550mW@220MHz	208LQFP	16kB/8kB

Table A-2 ARM Family Processors (continued)

2. *MIPS Family*

CPU Name	Manufact.	V_{CC}(V)	$V_{I/O}$(V)	Process(µm)	Temp(C)	MIPS @ MHz
TMP3912U	TOSHIBA	3.3	3.3	NA	0 / +70	@74
Galileo-3	GTI	NA	NA	NA	NA	NA
IDT79R3051/52	IDT	5	5	NA	0 / +85	35@40
IDT79R3081	IDT	5/3.3	5/3.3	NA	0 / +85	@50
79RC32364	IDT	3.3	3.3	NA	0 / +85	66@133
EZ4102	LSILOGIC	2.5/1.8	2.5/1.8	0.18	NA	68@85/@40@50
LR4102	LSILOGIC	2.5/1.8	3.3	0.18	0 / +70	68@85/40@50
TR4101	LSILOGIC	3.3	3.3	NA	NA	53@66
JADE	MIPS	2.5	2.5	0.25	NA	@150
MIPS324Kc	MIPS	2.5	2.5	0.18/0.25	NA	240@280/180@200
MIPS324Km	MIPS	2.5	2.5	0.18/0.25	NA	240@280/180@200
MIPS324Kp	MIPS	2.5	2.5	0.25	NA	@150
R4300i	MIPS	NA	NA	NA	NA	NA
μPD30111	NEC	3.3	3.3	NA	NA	@70
μPD705101	NEC	3.3	3.3	NA	-40 / +85	@33?
PR31500	PHILIPS	3.3	3.3	NA	0 / +70	@40
PR31100	PHILIPS	3.3	3.3	NA	0 / +70	@40
TMPR3901AF-70	TOSHIBA	3.3	3.3	NA	NA	74@70
TMPR3903AF	TOSHIBA	3.3	3.3	NA	-40 / +85	@40
TMPR3904AF-66	TOSHIBA	3.3	5	NA	NA	@66
TMPR3907F-66	TOSHIBA	3.3	3.3	NA	NA	@66
TMPR3912AU	TOSHIBA	3.3	3.3	NA	NA	@75
TMPR3912AU-92	TOSHIBA	3.3	3.3	NA	NA	@92
TMPR3912XB-92	TOSHIBA	3.3	3.3	NA	NA	@92
TMPR3922AU	TOSHIBA	2.7	3.3	NA	NA	@129
TMPR3922XB	TOSHIBA	2.7	3.3	NA	NA	@148
TMP1904AF	TOSHIBA	3.3	3.3	NA	NA	@20

Table A-3 MIPS Family Processors

MIPS Family (continued)

CPU Name	Bus (Addr/Data)	UART	Power	Packaging	Cache(I/D)
TMP3912U	32/32	None(Ext)	300mW	208QFP	4kB/1Kb
Galileo-3	16/16	1	NA	NA	NA
IDT79R3051/52	32/32	None	NA	84MQUAD	4(8)kB/2kB
IDT79R3081	32/32	None	1.2375W	84MQUAD	16(8)kB/4(8)kB
79RC32364	32/32	None	700mW	144TQFP	8kB/2kB
EZ4102	32/32?	1	NA	NA	$<$32kB
LR4102	32/32?	(SerialICE)	561mW	256PBGA	16kB/8kB
TR4101	32/32	None	NA	NA	NA
JADE	32/32	None	$<$2μW/MHz	NA	8kB
MIPS324Kc	32/32	None	1mW/MHz	NA	2-16kB
MIPS324Km	32/32	None	1mW/MHz	NA	0-16kB
MIPS324Kp	32/32	None	$<$1mW/MHz	NA	0-16kB
R4300i	32/32	None	1.8W	120PQFP	16KB/8kB
μPD30111	32/32	None	NA	224FBGA	16kB/8kB
μPD705101	32/32	1	760mW	160LQFP	4kB/4kB
PR31500	32/32	2	NA	208LQFP	4kB/1kB
PR31100	32/32	2	NA	208LQFP	4kB/1kB
TMPR3901AF-70	32/32	None	NA	NA	4kB/1kB
TMPR3903AF	32/32	None	800mW	208QFP	4kB/1kB
TMPR3904AF-66	32/32(16)	1	NA	208QFP	4kB/1kB
TMPR3907F-66	32/32(16)	1	NA	208QFP	4kB/1kB
TMPR3912AU	32/32	None	NA	208LQFP	4kB/1kB
TMPR3912AU-92	32/32	None	NA	208LQFP	4kB/1kB
TMPR3912XB-92	32/32	None	NA	217FBGA	4kB/1kB
TMPR3922AU	32/32	None	NA	208LQFP	16kB/8kB
TMPR3922XB	32/32	None	NA	217FBGA	16kB/8kB
TMP1904AF	32/32	1	NA	160QFP	4kB/1kB

Table A-4 MIPS Family Processors (continued)

3. *POWERPC Family*

CPU Name	Manufact.	V_{CC}(V)	$V_{I/O}$(V)	Process(µm)	Temp(C)	MIPS @ MHz
401GF	IBM	3.3	3.3	NA	NA	53@50
EM603/603e	IBM	3.3	3.3	0.2/0.26	0 / +105	23.9 SpecInt95@500
750	IBM	NA	NA	NA	NA	NA
602	IBM	3.3	3.3	0.5	0 / +105	48 SpecInt92@80
440	IBM	1.8	NA	0.18	NA	1000@550
604e	IBM	1.9	3.3	0.25	NA	@350
405GP	IBM	2.5	3.3/5	0.25	-40 / +85	252@200
403GCX	IBM	3.3	3.3/5	0.5	NA	NA
403GC	IBM	3.3	3.3/5	0.5	NA	@40
403GB	IBM	NA	NA	NA	NA	@28
403GA	IBM	NA	NA	NA	NA	@40
403GF	IBM	3.3	3.3/5	0.5	NA	@50
MPC821	MOTOROLA	3.3	3.3/5	NA	NA NA	53@40

Table A-5 POWERPC Family Processors

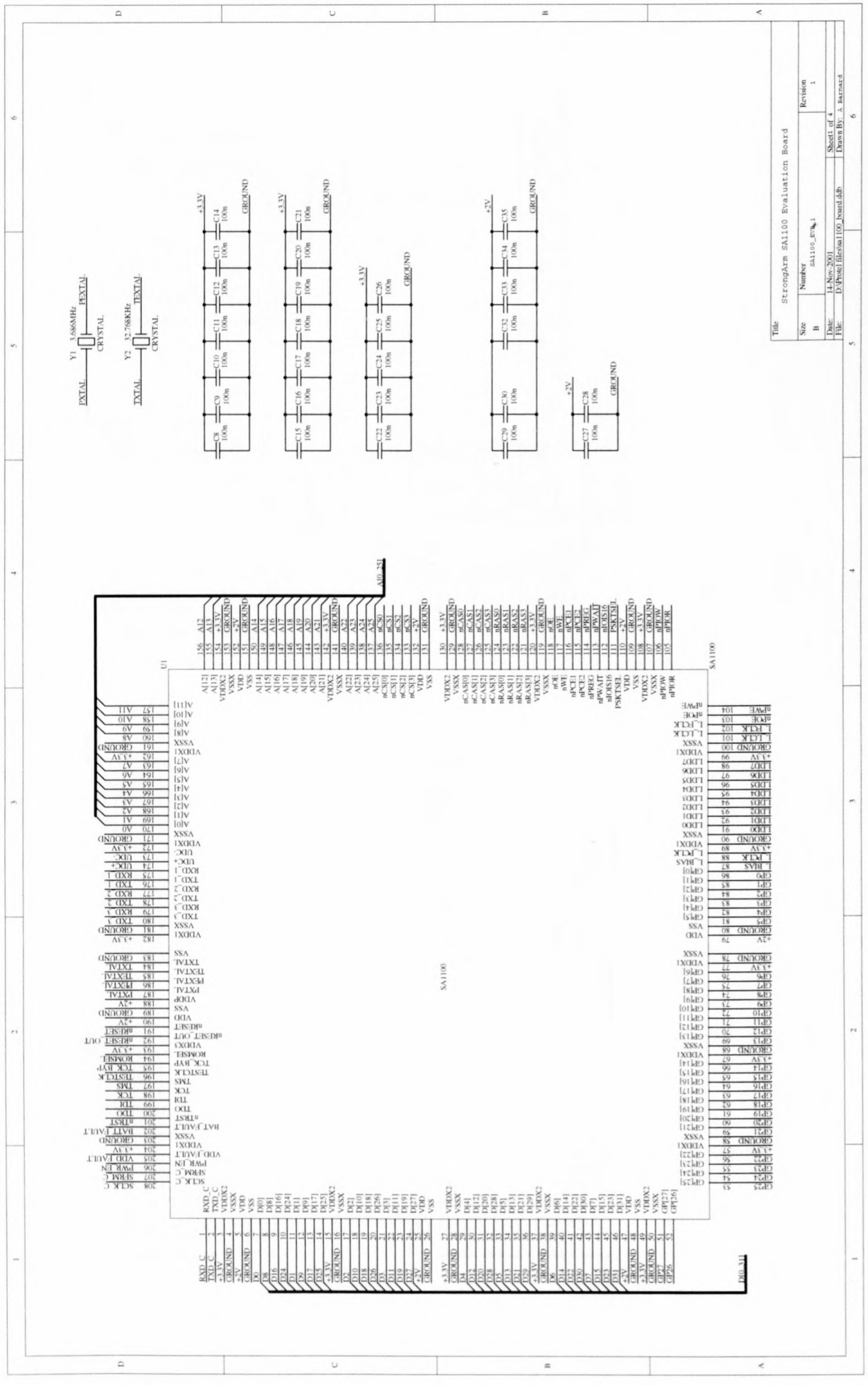
CPU Name	Bus (Addr/Data)	UART	Power	Packaging	Cache(I/D)
401GF	32/32	None	200mW	NA	2kB/1kB
EM603/603e	32/32(64)	None	NA	NA	16kB/16kB
750	32/32	None	4W	360BGA	32kB/32kB
602	32/32(64)	None	1.2W	144PQFP	4kB/4kB
440	32/32	None	1W	NA	$<$64kB/$<$64kB
604e	32/64	None	NA	NA	32kB/32kB
405GP	NA	NA	NA	NA	NA
403GCX	NA	NA	NA	NA	NA
403GC	NA	NA	NA	NA	NA
403GB	NA	NA	NA	NA	NA
403GA	NA	NA	NA	NA	NA
403GF	NA	NA	NA	NA	NA
MPC821	NA	NA	NA	NA	NA

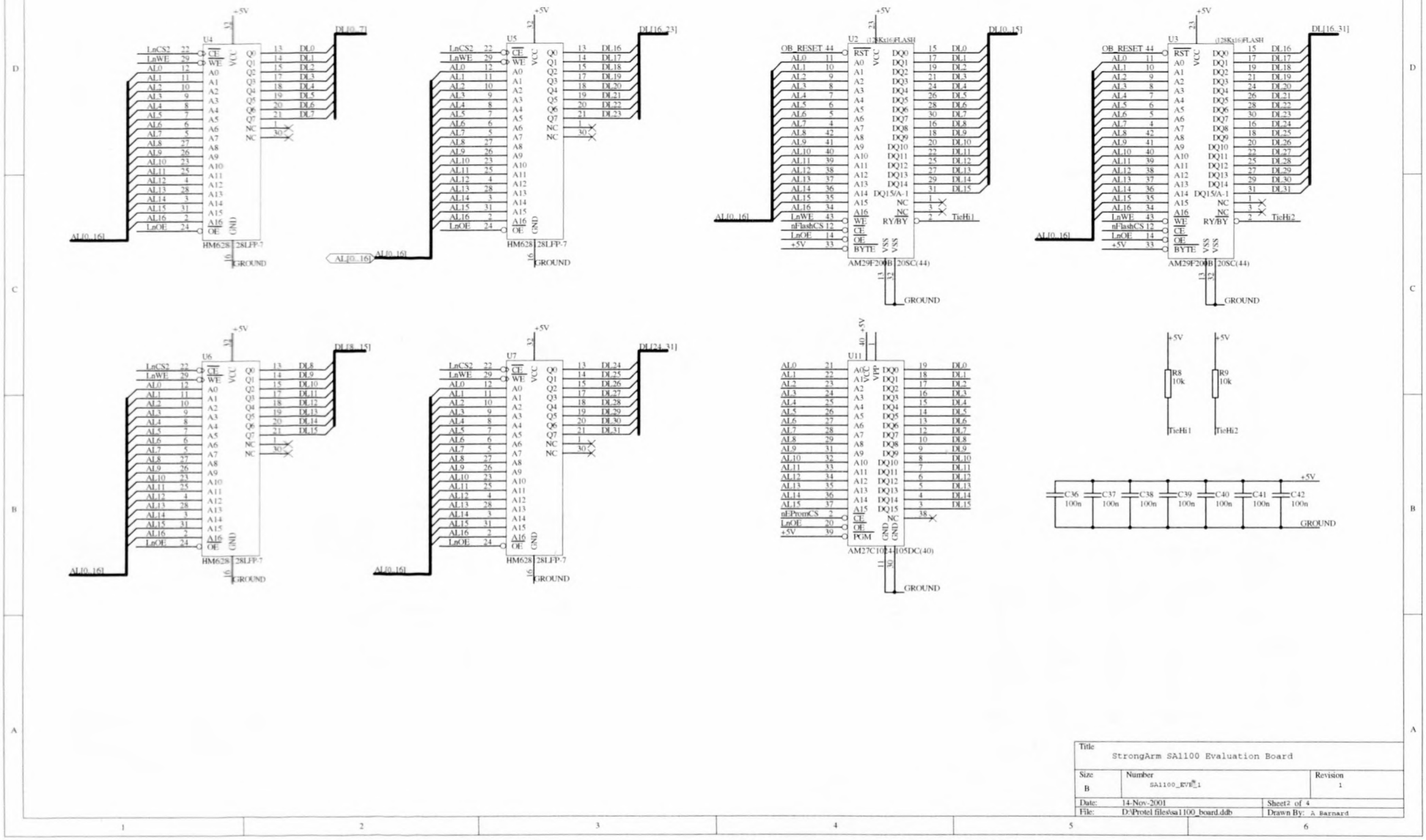
Table A-6 POWERPC Family Processors (continued)

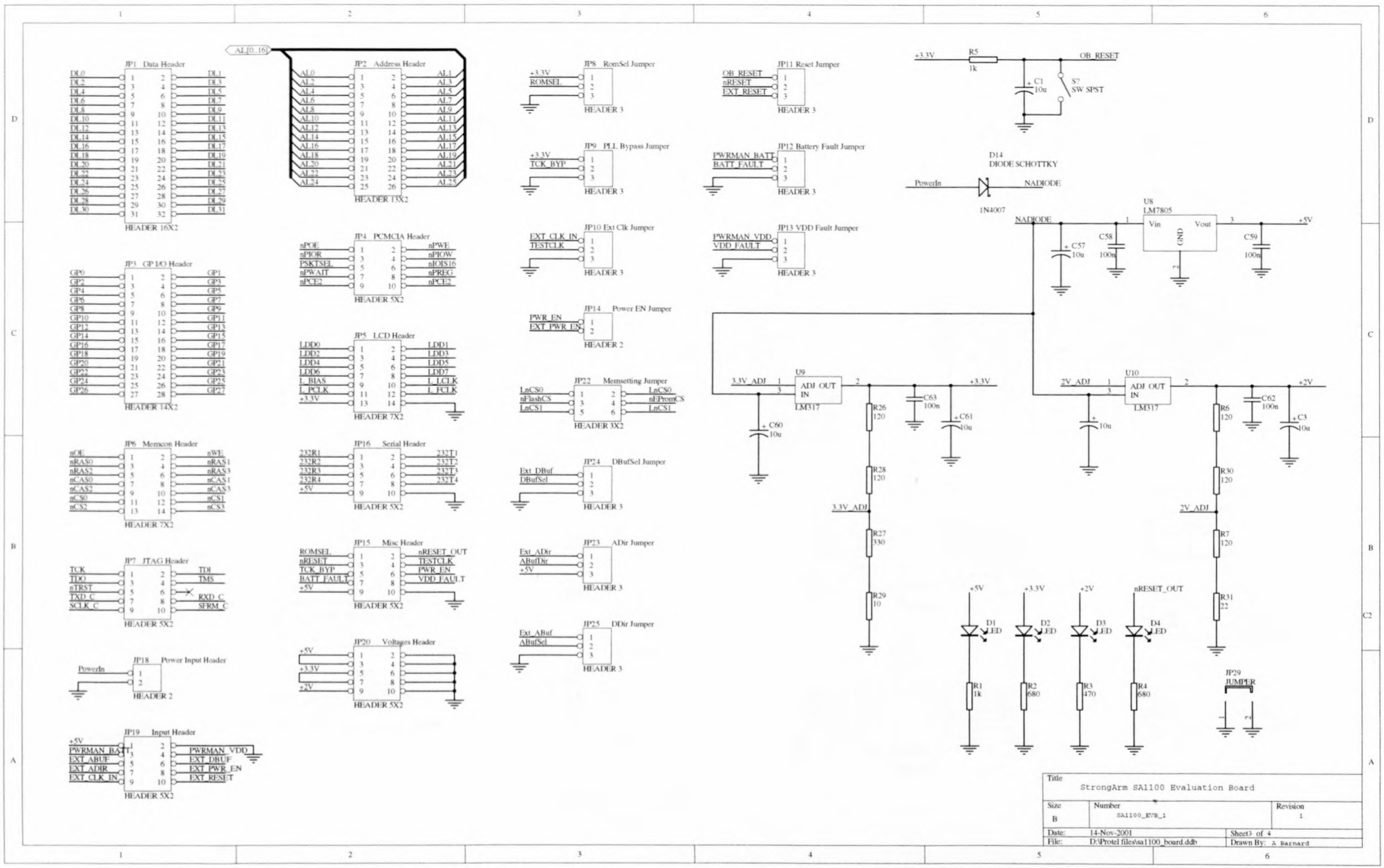
Appendix B

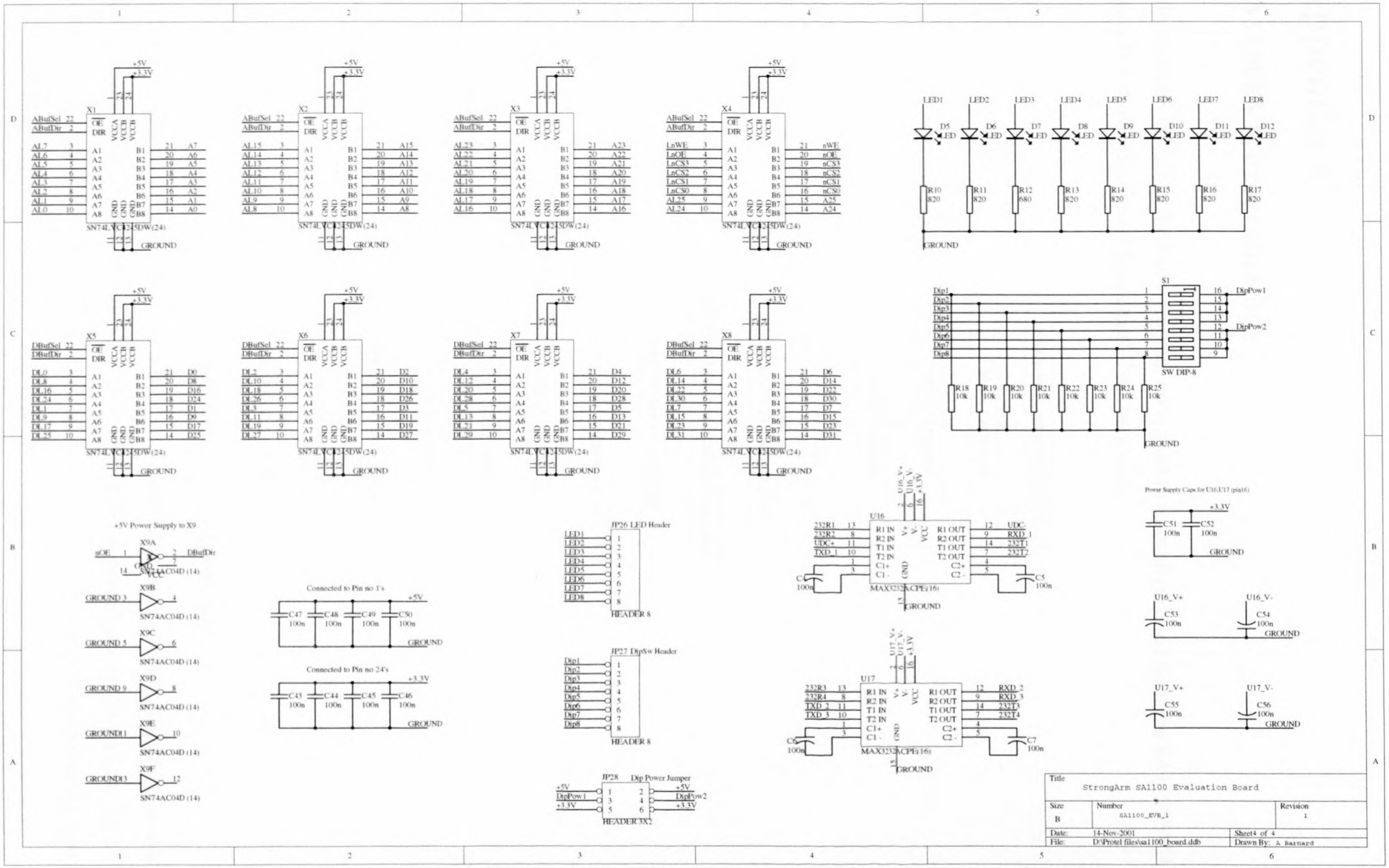
SA1100EVB Schematics

This appendix shows the schematics of the SA1100 Evaluation Board. The final layout of the board used a four-layer PCB. Dimensions of the final board are 11.7cm x 18.4cm.









Appendix C

SA1100EVB Program Code

This appendix describes the basic program code used in to configure and boot the SA1100EVB. Figure C-1 shows the overall flow chart of the boot code.

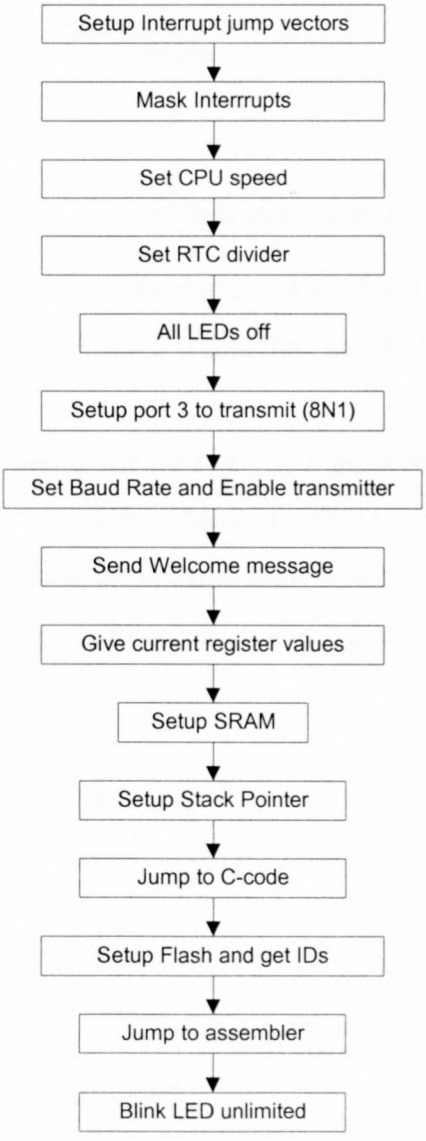


Figure C-1 Flow chart of overall boot code

The overview shown in Figure C-1 has one or two procedures that should be shown in greater detail to make their implementation clear. The first of these procedures is the `print_byte` procedure. This procedure is used to transmit the byte in register 0. Figure C-2 shows the flow diagram of this procedure.

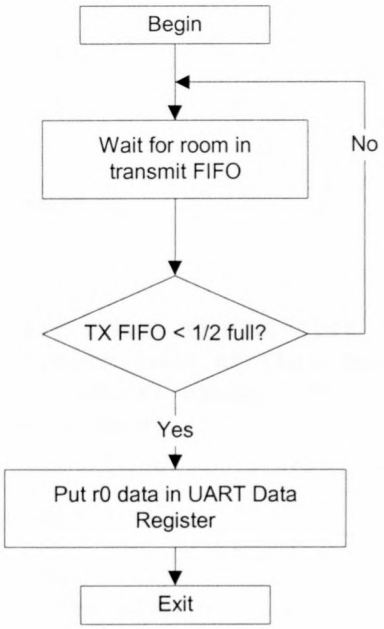


Figure C-2 Print_byte procedure flow chart

The `print_byte` procedure takes the data given to it in r0 and puts it in the data register of the UART when the transmit FIFO of the UART is less than half full.

To make transmission of strings easier, the `print_str` procedure is used. This procedure takes the string stored in memory, starting at the address in r0. The characters are then passed to the print_byte procedure and this process is repeated until a null character (0FFh) is reached. The flow diagram of the `print_str` procedure is shown in Figure C-3.

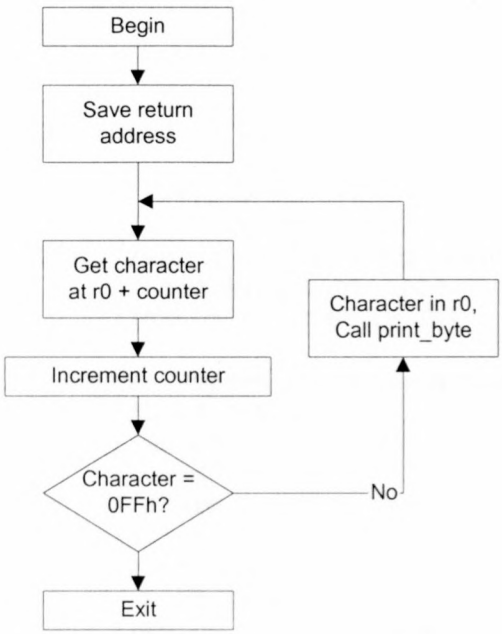


Figure C-3 Print_str procedure flowchart

The full assembly boot code is now listed.

```
//Filename: start.S
//Author:   Arno Barnard
//Date:     8/6/2001
//Version:  0.9

/*
 * start.S: SA1100EVB boot code
*/

/*
 * This is the SA1100EVB start code. The SA-1100 jumps to address 0x00000000
 * after a reset. A single branch code at this position which jumps
 * to a safe region to do the actual setup. All other vectors just point
 * to an endless loop for the moment.
 *
 * Version 0.9
 * Program performs in sequence :
 * - Masks Interrupts
 * - Sets up CPU Speed @ 60 MHz
 * - Transmit Hello World @ 57k6 nicely
 * - Setup SRAM
 * - Uses C-code to write out "Hello" through RS232
 * - Get the ST Flash ID from both ICs
 * - Blinky on unlimited
*/

.text

/* Jump vector table as in manual */
.globl _start
_start:     b     reset
      b     undefined_instruction
      b     software_interrupt
      b     abort_prefetch
      b     abort_data
      b     not_used
      b     irq
      b     fiq


//-----------------------------------------------------
reset:
/* Register addresses can be found in manual Appendix A */

      /* First, mask **ALL** interrupts */
      mov   r1, #0x90000000   //Load 0x90050000
      add   r1, r1, #0x50000  //which is Interrupt Control Registers Base
      mov   r2, #0x00         //Load with 0000 0000
      str   r2, [r1, #0x4]    //Store to ICMR (9-14)

      /* Switch the CPU to correct speed by writing the PPCR */
      mov   r1, #0x90000000   //Load 0x90020000
      add   r1, r1, #0x20000  //which is Power Manager Registers Base
      mov   r2, #0x00         //Load with 0000 0000 = 59 MHz
      str   r2, [r1, #0x14]   //Store in PPCR (9-35),(8-2)

      /* Switch the RTC to 1 seconds counting (untrimmed) */
      mov   r1, #0x90000000   //Load 0x90010000
      add   r1, r1, #0x10000  //which is RTC Register base
      mov   r2, #0x00008000   //Load to get = 1 Hz clock
      str   r2, [r1, #0x08]   //Store in RTTR (9-19)
```

```
//-----------------------------------------------------
led1on:
      /* Initialize the GPDR (GPIO Pin Direction Register) */
      /* in such a way that the LED is on an output port */

      /* load the GPIO base in r2 */
      mov   r2, #0x90000000
      add   r2, r2, #0x40000

      /*  Pins 1-8 is output for LED */
      mov   r1, #0x000000ff         //GPDR (9-4)
      str   r1, [r2, #0x04]

      /*  Put off all leds */
      mov   r1, #0x00000000
      str   r1, [r2, #0x08]



//-----------------------------------------------------
txstart:
      /* Try to transmit over the serial port 3. */
      mov   r1, #0x80000000         //Load 0x80050000
      add   r1, r1, #0x50000        //which is UART Registers
      mov   r2, #0xFF               //Load 1111 1111
      str   r2, [r1, #0x1C]         //Store in UTSR0 (11-141)

      /* Set the serial port to sensible defaults: */
      /* no break, no interrupts, no parity, 8 databits, 1 stopbit. */
      mov   r2, #0x00               //Load 0000 0000
      str   r2, [r1, #0x0C]         //Store in UTCR3 (11-136)
      mov   r2, #0x08               //Load 0000 1000
      str   r2, [r1, #0x00]         //Store in UTCR0 (11-133)


runner:
      /* Set BRD to 3, for a baudrate of 57k6 (Manual) */
      /* Set BRD to 5, for a baudrate of 38k4 (Manual) */
      /* Set BRD to 23, for a baudrate of 9k6 (Manual) */
      /* Set BRD to 191, for a baudrate of 1k2 (Manual) */
      mov   r1, #0x80000000         //Load 0x80050000
      add   r1, r1, #0x10000        //which is UART Registers
      mov   r2, #0x00               //Load 0000 0000
      str   r2, [r1, #0x04]         //Store in UTCR1 (MS Nibble) (11-134)
      mov   r2, #3                  //Load 1011 1111
      str   r2, [r1, #0x08]         //Store in UTCR2 (LS Nibble) (11-134)


// ****
      /* Enable the transmitter */
      mov   r2, #0x02               //Load 0000 0010
      str   r2, [r1, #0x0C]         //Store in UTCR3 (11-136)


      /* Send out a welcome message and status of system*/
      adr   r0, welcome_msg
      bl    print_str

      adr   r0, int_mask
      bl    print_str
      mov   r1, #0x90000000         //Load 0x90050000
      add   r1, r1, #0x50000        //is Interrupt Control Registers Base
      ldr   r0, [r1, #0x4]          //Store to ICMR (9-14)
      bl    print_hex
```

```
        adr   r0, speeder
        bl    print_str
        mov   r1, #0x90000000          //Load 0x90020000
        add   r1, r1, #0x20000         //which is Power Manager Registers Base
        ldr   r0, [r1, #0x14]          //Store to ICMR (9-14)
        bl    print_hex

        adr   r0, dirled
        bl    print_str
        mov   r1, #0x90000000
        add   r1, r1, #0x40000
        ldr   r0, [r1, #0x04]
        bl    print_hex

        adr   r0, uart_baud
        bl    print_str
        mov   r1, #0x80000000          //Load 0x80050000
        add   r1, r1, #0x50000         //which is UART Registers (Port3)
        ldr   r0, [r1, #0x04]
        bl    print_hex
        ldr   r0, [r1, #0x08]
        bl    print_hex

        adr   r0, linefeed
        bl    print_str

//-----------------------------------------------------
ram_main:
        /* Setup SRAM */
        adr   r0, sram_msg
        bl    print_str
        mov   r1, #0xA0000000          //Base adress for memory control
        mov   r2, #0x00003F00          //Get MSC1 register value make top 16
bits zero
        add   r2, r2, #0x0F9
        str   r2, [r1, #0x14]          //Write to MSC1 (10-10)
        mov   r0, r2
        bl    print_hex
        adr   r0, linefeed
        bl    print_str

//-----------------------------------------------------
        /* Set up the stack pointer */
        mov   r2, #0x10000000
        add   r2, r2, #0x10000
        sub   sp, r2, #0x04

        /* Arg 0 of the C code is the start of the block it can use; */
        /* arg 1 is the size of that block. */
        mov   r0, #0x10000000          // Start of SRAM
        mov   r1, #0x10000

        /* Jump to the C code */
jump_to_c:
        bl    main

//-----------------------------------------------------

infinite:
        b     infinite


blinky:
```

```
        /* The old blinker */
        mov     r2, #0x90000000
        add     r2, r2, #0x40000
        mov     r1, #0x00000001

        /* Set GPIO pin as ouput for LED, in GPDR register */
        str   r1, [r2, #0x04]

old_led_on:
        /* turn on the LED by writing the GPSR (GPIO Pin output Set
        Register) */
        str   r1, [r2, #0x08]
        mov   r4, #0x10000
loop1:
        subs  r4, r4, #1
        bne   loop1

old_led_off:
        /* turn off the LED by writing the GPCR (GPIO Pin output Clear
        Register) */
        str   r1, [r2, #0x0c]

        mov     r4, #0x10000
loop2:
        subs    r4, r4, #1
        bne   loop2

        /* and loop forever */
        mov   r1, #0x90000000
        add   r1, r1, #0x10000
        ldr   r0, [r1, #0x04]
        bl    print_hex
        adr   r0, carriage
        bl    print_str

        b     blinky

        b     reset


//-------------------------------------------------------
        /* Send dying message and goes into infinite loop */
        /* The end of all ends! */
die:
        adr   r0, die_msg
        bl    print_str
dieloop:
        b     dieloop

.align 4
linefeed:
        .string "\r\n"
.align 4
int_mask:
        .string "\r\n (ICMR) : "
.align 4
speeder:
        .string "\r\n (PPCR) : "
.align 4
dirled:
        .string "\r\n (GPDR) : "
.align 4
uart_baud:
        .string "\r\n (UTCR1=MSb , UTCR2=LSb) : "
```

```
.align 4
timed:
      .string "\r\nTime for test : "
.align 4
welcome_msg:
      .string "SA_Bootter(v0.9) running on SA1100EVB(v0.1)\r\n"
.align 4


//-------------------------------------------------------
undefined_instruction:

/* Initialize the GPDR (GPIO Pin Direction Register) */
/* in such a way that the LED is on an output port */

      /* Send out a message */
      adr   r0, undefi_inst_msg
      bl    print_str

      b     die

//-------------------------------------------------------
software_interrupt:

/* Initialize the GPDR (GPIO Pin Direction Register) */
/* in such a way that the LED is on an output port */

      /* Send out a message */
      adr   r0, softw_intr_msg
      bl    print_str

      b     die

//-------------------------------------------------------
abort_prefetch:

/* Initialize the GPDR (GPIO Pin Direction Register) */
/* in such a way that the LED is on an output port */

      /* Send out a message */
      adr   r0, abort_pre_msg
      bl    print_str

      b     die

//-------------------------------------------------------
abort_data:

/* Initialize the GPDR (GPIO Pin Direction Register) */
/* in such a way that the LED is on an output port */

      /* Send out a message */
      adr   r0, abort_dta_msg
      bl    print_str

      b     die

//-------------------------------------------------------
not_used:

/* Initialize the GPDR (GPIO Pin Direction Register) */
/* in such a way that the LED is on an output port */

      /* Send out a  message */
```

```
        adr   r0, not_used_msg
        bl    print_str

        b     die

//-------------------------------------------------------
irq:

/* Initialize the GPDR (GPIO Pin Direction Register) */
/* in such a way that the LED is on an output port */

        /* Send out a message */
        adr   r0, irq_msg
        bl    print_str

        b     die

//-------------------------------------------------------
fiq:

/* Initialize the GPDR (GPIO Pin Direction Register) */
/* in such a way that the LED is on an output port */

        /* Send out a message */
        adr   r0, fiq_msg
        bl    print_str

        b     die

//-------------------------------------------------------

.align 4
dot:
        .string "."
.align 4
carriage:
        .string "\r"
.align 4
sram_msg:
        .string "\r\nSRAM Setup: MSC1 = "
.align 4
numbererr:
        .string "number of errors : "
.align 4
reading:
        .string "Reading from all SRAM addresses..."
.align 4
wrting1:
        .string "Writing 0xF0F0 F0F0 to all SRAM addresses..."
.align 4
wrting2:
        .string "Writing 0x0F0F 0F0F to all SRAM addresses..."
.align 4
undefi_inst_msg:
        .string "Undefined Instruction!\r\n"
.align 4
softw_intr_msg:
        .string "Software Interrupt!\r\n"
.align 4
abort_pre_msg:
        .string "Prefetch Abort!\r\n"
.align 4
abort_dta_msg:
        .string "Data Abort!\r\n"
```

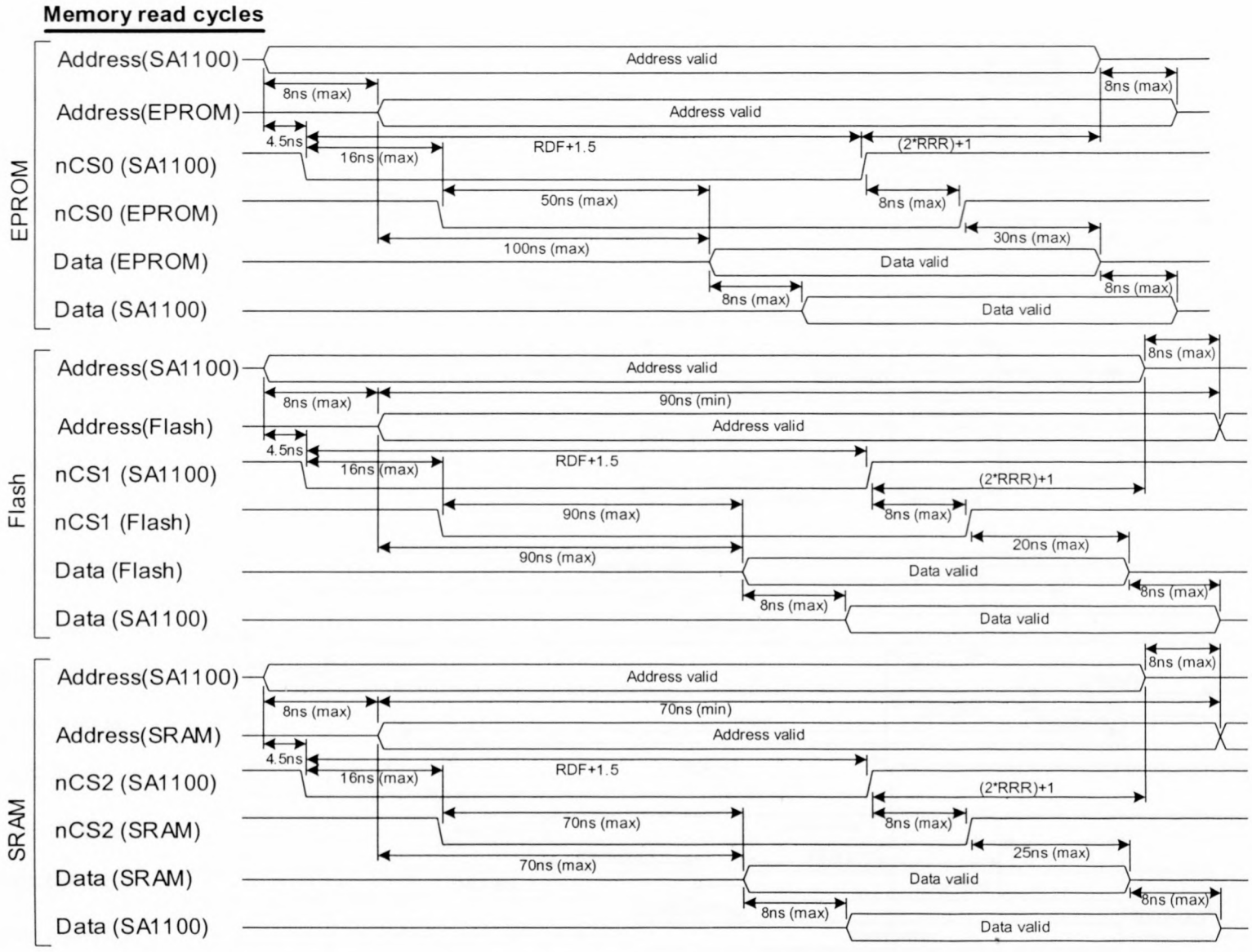
```
.align 4
not_used_msg:
      .string "Exception Not Used!\r\n"
.align 4
irq_msg:
      .string "IRQ Happened!\r\n"
.align 4
fiq_msg:
      .string "FIQ Happened!\r\n"
.align 4
die_msg:
      .string "I just died!! Please help me master!! ...\r\n\r\n"
.align 4


.globl print_str
.globl print_hex
.globl print_byte
//-------------------------------------------------------
      /* Subroutine that sends a string over the serial port */
      /* The address of the string should be in r0 */
print_str:
      /* Save the return address */
      mov   r13, r14
      mov   r2, r0
prs1:
      ldrsb r0, [r2]
      add   r2, r2, #0x01
      ands  r0, r0, #0xFF
      beq   prs2
      bl    print_byte
      b     prs1
prs2:
      /* Return */
      mov   pc, r13



//-------------------------------------------------------
      /* Subroutine to send a hex word over the serial port */
      /* The hex value is in r0 */
print_hex:
      mov   r13, r14
      mov   r2, r0
      mov   r3, #0x08
      mov   r0, #0x30
      bl    print_byte
      mov   r0, #0x78
      bl    print_byte
prh1:
      and   r0, r2, #0xF0000000
      mov   r0, r0, lsr #28
      add   r0, r0, #0x30
      cmp   r0, #0x3A
      addge r0, r0, #0x07
      bl    print_byte
      mov   r2, r2, lsl #4
      subs  r3, r3, #0x01
      bne   prh1

      mov   pc, r13
```

```
//-------------------------------------------------------
      /* Subroutine that sends a byte over the serial port 3. */
      /* The byte is in r0 */
print_byte:
      /* Wait for room in the tx fifo */
      mov     r1, #0x80000000
      add     r1, r1, #0x50000

      ldr     r1, [r1, #0x1C]
      ands    r1, r1, #0x01
      beq   print_byte

      mov     r1, #0x80000000
      add   r1, r1, #0x50000

      str   r0, [r1, #0x14]
      mov   pc, r14

/* THE END */
```

Appendix D

Timing analyses

This appendix shows the diagrams of the timing analyses done for the memory system and processor interfaces. The main concern was that the delays caused by the inclusion of the 74LVX4245 level shifters could create timing problems in the system. Read and write cycle timing diagrams of the SA1100 processor and of the EPROM, SRAM and Flash memories were compared. The resultant timing diagrams (including the delays due to the level shifters) are shown in Figure D-1 and Figure D-2. From the timing analyses, the conclusion is drawn that the timing problems can easily be solved by setting the RDF, RDN and RRR fields in the relevant registers to the desired value.



Note : nOE signal changes together with nCSx in each case, only nCSx is shown since it has the longer timing.

Figure D-1 Memory read cycle timing analyses

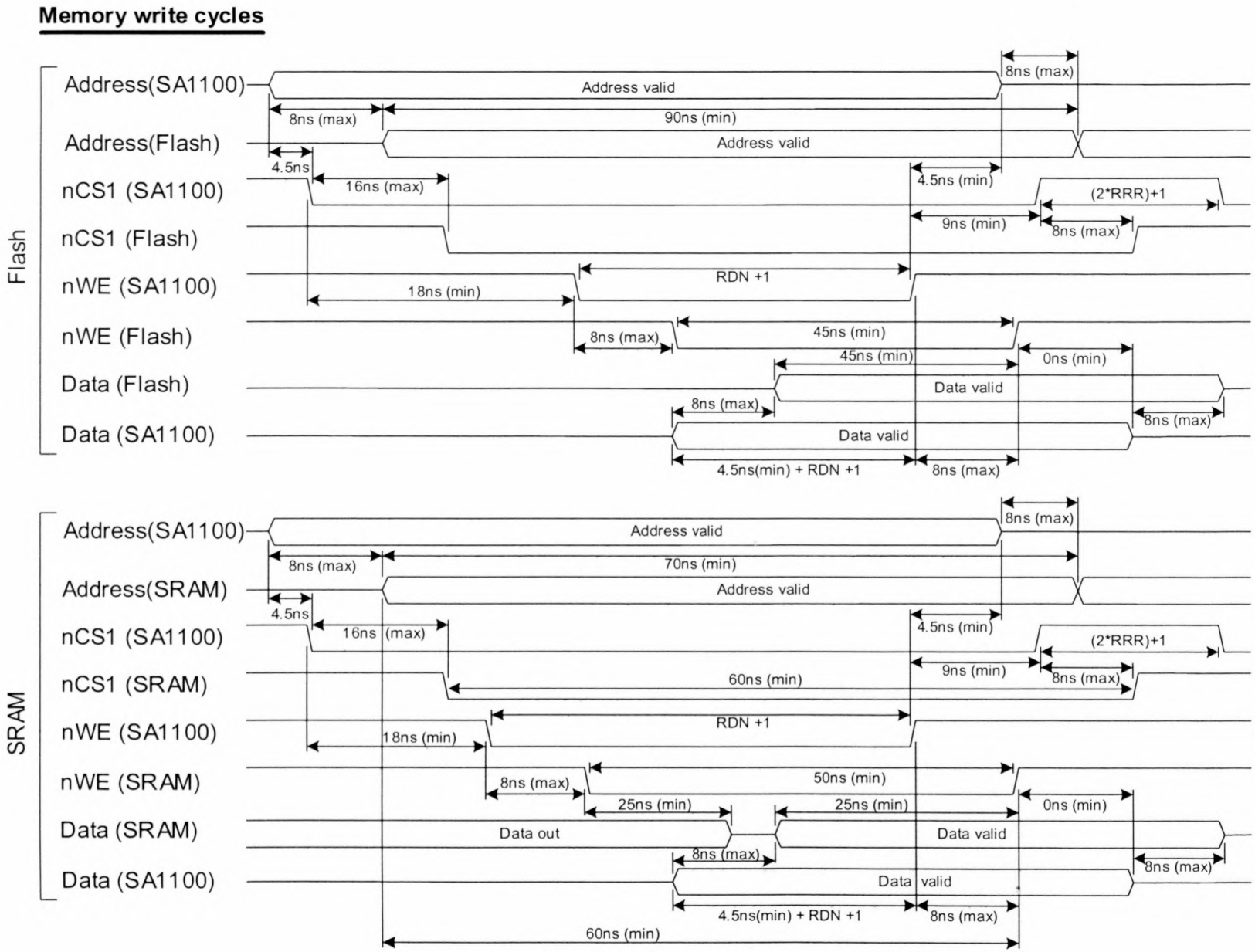


Figure D-2 Memory write cycle timing analyses